EDGEWOOD

RESEARCH, DEVELOPMENT & ENGINEERING CENTER

U.S. ARMY CHEMICAL AND BIOLOGICAL DEFENSE COMMAND

ERDEC-TR-329

LIGHT SCATTERING FROM NONCONCENTRIC SPHERES

Dat Ngo
Steve Christesen

RESEARCH AND TECHNOLOGY DIRECTORATE

Gorden Videen

U.S. ARMY RESEARCH LABORATORY
White Sands Missile Range, NM 88002-5001

May 1996

CUM SCIENTIA DEFENDIMUS

Aberdeen Proving Ground, MD 21010-5423

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE 1996 May	3. REPORT TYPE AND DATES COVERED Final, 93 Sep - 96 Apr

4. TITLE AND SUBTITLE
Light Scattering from Nonconcentric Spheres

5. FUNDING NUMBERS
PR-10161102A71A

6. AUTHOR(S)
Ngo, Dat, Christesen, Steve (ERDEC); Videen, Gordon (ARL - WSMR Site)

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
DIR, ERDEC, ATTN: SCBRD-RTE, APG, MD 21010-5423
DIR, ARL, WSMR, NM 88002-5001

8. PERFORMING ORGANIZATION REPORT NUMBER
ERDEC-TR-329

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Atmospheric aerosols often contain small inclusions of insoluble material which can affect remote sensing techniques that rely on the scattering or fluorescence for detection, and identification. Thus the exact solution for the scattering from a sphere with a nonconcentric spherical inclusion is derived using an extension of Mie theory. The boundary conditions at the surfaces of both spheres are satisfied by representing the fields emanating from the center of one sphere as a summation of vector harmonics emanating from the center of the other sphere. The exact solution is shown to reduce to the coated sphere solution if the inclusion and host spheres are concentric, and to the Mie theory solution if the refractive indices of the inclusion and host are the same. We then examine the sensitivity of the scattering to various system parameters.

14. SUBJECT TERMS
Aerosol research   Light scattering

15. NUMBER OF PAGES
78

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Blank

PREFACE

The work described in this report was authorized under Project No. 10161102A71A, Smoke Technology. This work was started in September 1993 and completed in April 1996.

The use of trade or manufacturers' names in this report does not constitute an official endorsement of any commercial products. This report may not be cited for purposes of advertisement.

This report has been approved for release to the public. Registered users should request additional copies from the Defense Technical Information Center; unregistered users should direct such requests to the National Technical Information Center.

Acknowledgments

The authors would like to thank Ronald Pinnick and Peter Chylek for their support and discussion. They would also like to thank Roger Combs and Dennis Roser for helpful suggestions and moral support.

Blank

CONTENTS

FIGURES

TABLES

LIGHT SCATTERING FROM NONCONCENTRIC SPHERES

1 INTRODUCTION

Atmospheric aerosols often contain small inclusions of insoluble material which can affect their scattering properties. These inclusions can be found anywhere within the host aerosols. Chemical and biological agents can also exist in the atmosphere as multi component aerosols. For example, bacterial agents can be dispersed as a liquid suspension where the spore or cell is embedded in a water droplet. In the 1950's, the use of carrier dusts for disseminating chemical agents was tested.[1],[2] Although unsuccessful, the goal of this work was to modify the physical characteristics of the chemical agent by use of an inert dust as a carrier to obtain more efficient aerosol dissemination. It is also assumed that any inclusion or encapsulation can directly affect the optical properties of the aerosol, and hence its scattering characteristics. This influence has consequences for remote sensing techniques that rely on the aerosol scattering or fluorescence for detection and identification. Therefore, the purpose of this report is to show the theoretical derivation of a host sphere containing one spherical inclusion arbitrarily located within the host.

Until recently, theoretical predictions of scattering for a host sphere containing a nonconcentrically positioned smaller sphere was not feasible. Most experimentalists had to be content with the concentric sphere model,[3]–[5] i.e., the smaller sphere sharing the same origin as the host sphere. Some made use of the T-matrix method [6]–[8] to examine a radially inhomogeneous sphere, and some even dabbled with the effective medium approximations.[9] It was not until Friedman and Russek [10] published the addition theorem for spherical waves that the solution to the nonconcentric problem was even realizable. However, Friedman and Russek made some errors in their derivations. Stein [11] (and later Cruzan [12]) found the errors and came up with the correct translation addition theorem for spherical wave functions. After that, the solution was attainable. Fikioris and Uzunoglu [13] were the first to obtain a mathematical expression for the scattering of a nonconcentric sphere within a host sphere, followed by Borghese *et al.*[14] Kirk Fuller derived the relations for the same problem using an order of scattering approach.[15] In the following pages we derive the general expressions for the scattering from a system composed of a sphere containing a nonconcentric spherical inclusion.

2 THE SOLUTION

The scattering solution of a sphere containing many inclusions is difficult to solve, and even more difficult to implement because of the limitation of present day computers. For this reason, we choose a simple model having one spherical inclusion arbitrarily placed inside a larger sphere.

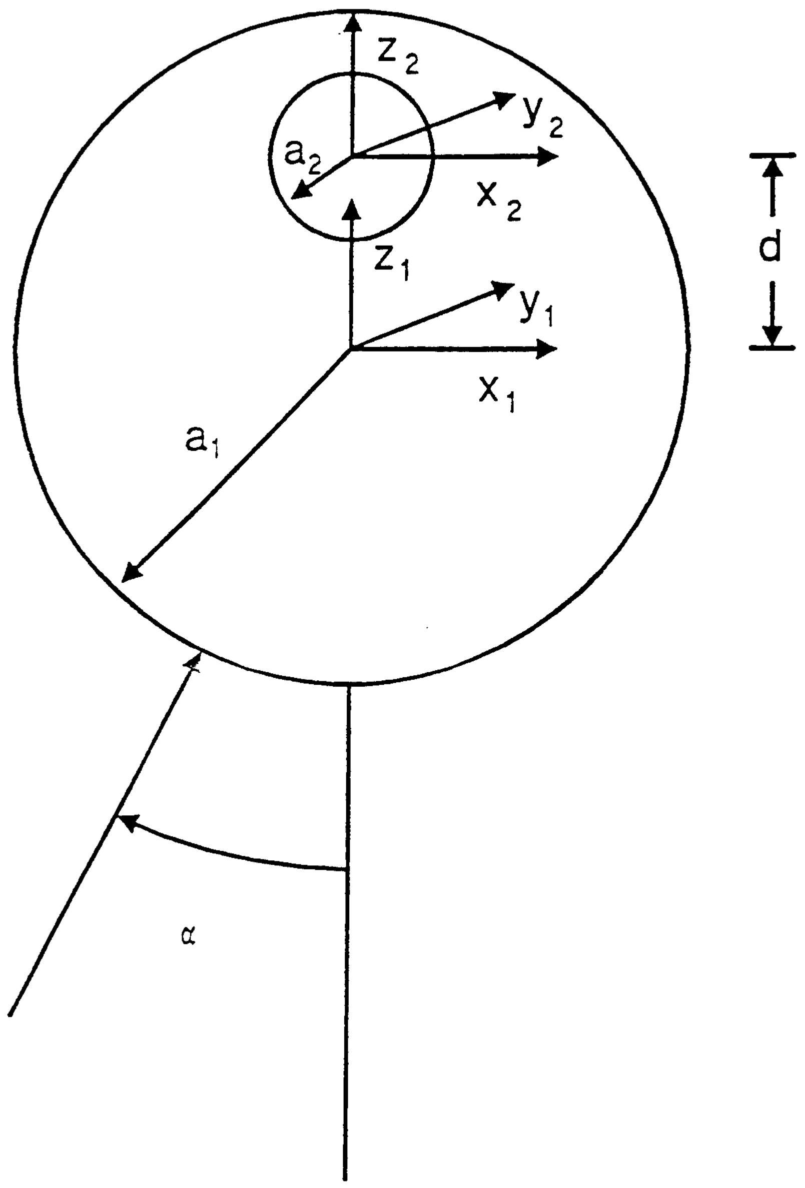


Figure 1. Geometry of the nonconcentric sphere scattering system. A host sphere of radius a_1 and complex refractive index $\tilde{m}_1$ is centered on the $x_1y_1z_1$ coordinate system. The inclusion sphere of radius a_2 and complex refractive index $\tilde{m}_2$ is centered on the $x_2y_2z_2$ coordinate system at a position $x_1 = 0$. $y_1 = 0$. $z_1 = d$. The incident radiation is a plane wave traveling in the $x - z$ plane, oriented at angle α with respect to the z axis.

The geometry of the scatterer is shown in Figure 1. The larger sphere (the host) of radius a_1 and refractive index m_1 completely encloses the smaller sphere (the inclusion) of radius a_2 and refractive index m_2. The center of the host sphere is situated on the x_1, y_1, z_1 coordinate system. The inclusion sphere is centered on the x_2, y_2, z_2 coordinate system at a position $x_1 = 0$, $y_1 = 0$, $z_1 = d$. Since the inclusion sphere is centered on the z_1 axis, to make the solution completely general, we require the incident radiation to impinge upon the system at an arbitrary angle α. The problem of a sphere within a sphere can be solved by simultaneously satisfying all the boundary conditions at the surfaces of the two spheres.

Figure 2 shows the physical representation of the fields. Note that we use circular arcs to represent spherical traveling waves (spherical Bessel functions of the 3rd and 4th kinds) and parallel lines to represent plane waves. The boundary conditions at the surfaces of both spheres are satisfied by representing the fields emanating from the center of one sphere as a summation of vector harmonics emanating from the center of the other sphere. We use the vector spherical harmonics which have the following form:

$$\begin{aligned}
\mathbf{M}^{(\rho)}_{nm,j} &= \hat{\theta}_j \left[\frac{im}{\sin\theta_j} z_n^{(\rho)}(kr_j) \tilde{P}_n^m(\cos\theta_j) e^{im\varphi_j} \right] - \\
&\quad \hat{\varphi}_j \left[z_n^{(\rho)}(kr_j) \frac{d}{d\theta_j} \tilde{P}_n^m(\cos\theta_j) e^{im\varphi_j} \right],
\end{aligned}$$

$$\begin{aligned}
\mathbf{N}^{(\rho)}_{nm,j} &= \hat{r}_j \left[\frac{1}{kr_j} z_n^{(\rho)}(kr_j) n(n+1) \tilde{P}_n^m(\cos\theta_j) e^{im\varphi_j} \right] + \\
&\quad \hat{\theta}_j \left[\frac{1}{kr_j} \frac{d}{dr_j} \left(r_j z_n^{(\rho)}(kr_j) \right) \frac{d}{d\theta_j} \tilde{P}_n^m(\cos\theta_j) e^{im\varphi_j} \right] + \\
&\quad \hat{\varphi}_j \left[\frac{1}{kr_j} \frac{d}{dr_j} \left(r_j z_n^{(\rho)}(kr_j) \right) \frac{im}{\sin\theta_j} \tilde{P}_n^m(\cos\theta_j) e^{im\varphi_j} \right],
\end{aligned} \tag{1}$$

where the index j corresponds to the coordinate system used (j = 1,2) and $z_n^{(\rho)}(kr_j)$ are the spherical Bessel functions of the first, second, third, or fourth kind ($\rho = 1, 2, 3, 4$), and

$$\tilde{P}_n^m(\cos\theta_j) = \sqrt{\frac{(2n+1)(n-m)!}{2(n+m)!}} P_n^m(\cos\theta_j), \tag{2}$$

where $P_n^m(\cos\theta_j)$ are the associated Legendre polynomials.

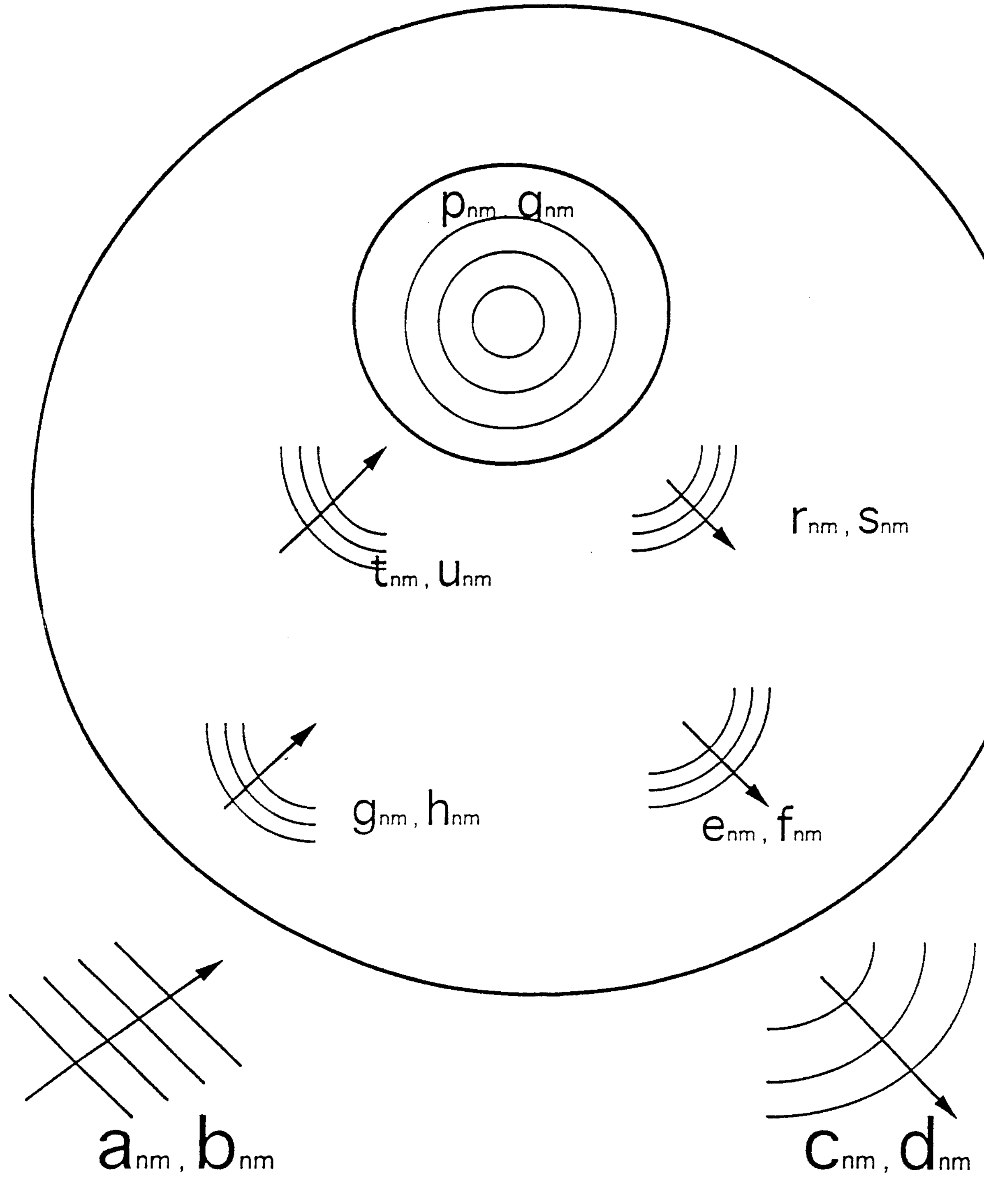


Figure 2. Physical representation of the fields on the nonconcentric sphere system.

2.1 Host Sphere

We now examine the fields which strike the host sphere surface ($j = 1$). We consider an arbitrary field incident on the system which can be expanded using the spherical Bessel functions of the first kind, $j_n(kr_1)$. We will examine the specific cases of plane waves polarized perpendicular and parallel to the y axis later. The incident electric field may be expanded as

$$\mathbf{E}^1_{\mathbf{inc}} = \sum_{n=0}^{\infty} \sum_{m=-n}^{n} a_{nm}\mathbf{M}^{(1)}_{nm,1} + b_{nm}\mathbf{N}^{(1)}_{nm,1}. \tag{3}$$

Similarly, the scattered electric field may be expanded using the spherical Bessel functions of the third kind, $h_n^{(1)}(kr_1)$:

$$\mathbf{E}^1_{\mathbf{sca}} = \sum_{n=0}^{\infty} \sum_{m=-n}^{n} c_{nm}\mathbf{M}^{(3)}_{nm,1} + d_{nm}\mathbf{N}^{(3)}_{nm,1}. \tag{4}$$

The fields near the boundary $|d| < r_1 < a_1$ inside the sphere may be expanded into incoming and outgoing spherical waves using spherical Bessel functions of the fourth kind $h_n^{(2)}(k_1 r_1)$ and third kind $h_n^{(1)}(k_1 r_1)$:

$$\mathbf{E}^1_{\mathbf{sph}} = \sum_{n=0}^{\infty} \sum_{m=-n}^{n} e_{nm}\mathbf{M}^{(3)}_{nm,1} + f_{nm}\mathbf{N}^{(3)}_{nm,1} + g_{nm}\mathbf{M}^{(4)}_{nm,1} + h_{nm}\mathbf{N}^{(4)}_{nm,1}. \tag{5}$$

The application of boundary conditions at the host sphere surface for the above three equations yields two sets of equations:

$$a_{nm}k_1\psi_n(ka_1) + c_{nm}k_1\xi_n^{(1)}(ka_1) = e_{nm}k\xi_n^{(1)}(k_1a_1) + g_{nm}k\xi_n^{(2)}(k_1a_1), \tag{6}$$

$$a_{nm}\psi'_n(ka_1) + c_{nm}\xi_n'^{(1)}(ka_1) = e_{nm}\xi_n'^{(1)}(k_1a_1) + g_{nm}\xi_n'^{(2)}(k_1a_1), \tag{7}$$

$$b_{nm}\psi_n(ka_1) + d_{nm}\xi_n^{(1)}(ka_1) = f_{nm}\xi_n^{(1)}(k_1a_1) + h_{nm}\xi_n^{(2)}(k_1a_1), \tag{8}$$

$$b_{nm}k_1\psi'_n(ka_1) + d_{nm}k_1\xi_n'^{(1)}(ka_1) = f_{nm}k\xi_n'^{(1)}(k_1a_1) + h_{nm}k\xi_n'^{(2)}(k_1a_1), \tag{9}$$

where $\psi_n(r)$ and $\xi_n^{(q)}(r)$ (q = 1,2) are the Riccati-Bessel functions defined by

$$\psi_n(r) = rj_n(r) \text{ and } \xi_n^{(q)}(r) = rh_n^{(q)}(r), \tag{10}$$

and the primes denote derivatives with respect to the argument.

2.2 Inclusion Sphere

Next we examine the fields which strike the inclusion sphere surface ($j = 2$). The fields inside this sphere may be expressed using spherical Bessel functions of the first kind $j_n(k_2 r_2)$:

$$\mathbf{E}^2_{\text{int}} = \sum_{n=0}^{\infty} \sum_{m=-n}^{n} p_{nm}\mathbf{M}^{(1)}_{nm,2} + q_{nm}\mathbf{N}^{(1)}_{nm,2}. \tag{11}$$

The fields near the boundary outside the sphere may be expressed into incoming and outgoing spherical waves using spherical Bessel functions of the fourth kind $h^{(2)}_n(k_1 r_2)$ and third kind $h^{(1)}_n(k_1 r_2)$:

$$\mathbf{E}^2_{\text{ext}} = \sum_{n=0}^{\infty} \sum_{m=-n}^{n} r_{nm}\mathbf{M}^{(3)}_{nm,2} + s_{nm}\mathbf{N}^{(3)}_{nm,2} + t_{nm}\mathbf{M}^{(4)}_{nm,2} + u_{nm}\mathbf{N}^{(4)}_{nm,2}. \tag{12}$$

Applying boundary conditions at the sphere surface yields two sets of equations:

$$p_{nm}k_1\psi_n(k_2 a_2) = r_{nm}k_2\xi^{(1)}_n(k_1 a_2) + t_{nm}k_2\xi^{(2)}_n(k_1 a_2), \tag{13}$$

$$p_{nm}\psi'_n(k_2 a_2) = r_{nm}\xi'^{(1)}_n(k_1 a_2) + t_{nm}\xi'^{(2)}_n(k_1 a_2), \tag{14}$$

$$q_{nm}\psi_n(k_2 a_2) = s_{nm}\xi^{(1)}_n(k_1 a_2) + u_{nm}\xi^{(2)}_n(k_1 a_2), \tag{15}$$

$$q_{nm}k_1\psi'_n(k_2 a_2) = s_{nm}k_2\xi'^{(1)}_n(k_1 a_2) + u_{nm}k_2\xi'^{(2)}_n(k_1 a_2). \tag{16}$$

We can eliminate the interior field coefficients (p_{nm} and q_{nm}) to find relationships for the exterior field coefficients. After a little bit of algebra, we have:

$$r_{nm} = t_{nm}\frac{k_1\xi'^{(2)}_n(k_1 a_2)\psi_n(k_2 a_2) - k_2\xi^{(2)}_n(k_1 a_2)\psi'_n(k_2 a_2)}{k_2\xi^{(1)}_n(k_1 a_2)\psi'_n(k_2 a_2) - k_1\xi'^{(1)}_n(k_1 a_2)\psi_n(k_2 a_2)} = Q^r_n t_{nm}. \tag{17}$$

$$s_{nm} = u_{nm}\frac{k_2\xi'^{(2)}_n(k_1 a_2)\psi_n(k_2 a_2) - k_1\xi^{(2)}_n(k_1 a_2)\psi'_n(k_2 a_2)}{k_1\xi^{(1)}_n(k_1 a_2)\psi'_n(k_2 a_2) - k_2\xi'^{(1)}_n(k_1 a_2)\psi_n(k_2 a_2)} = Q^s_n u_{nm}, \tag{18}$$

where Q^r_n and Q^s_n are the Q factors which contain information about the inclusion sphere such as its size and refractive index.

2.3 Fields Interior to the Host Sphere

The fields in the interior of the host sphere are expressed by equations 6-9, while the fields exterior to the inclusion sphere are expressed by equations 17 and 18. At first glance, it is tempting to merely equate those coefficients, but that would be a mistake since the vector spherical harmonics expressed by those equations are centered about two different coordinate systems. This is where the importance of the translational addition theorem comes into play.

Stein[12] and Cruzan[13] have derived translation addition theorems for vector spherical wave functions which can be used to express the coefficients e_{nm}, f_{nm}, g_{nm}, and h_{nm}in terms of the coefficients r_{nm}, s_{nm}, t_{nm}, and u_{nm}. And the way we have set up our geometry for the scattering system now comes in handy. Because we require that the inclusion be centered along the z-axis, the two origins are separated only by a displacement in z. Hence, there is no rotation. Thus, for a translation along the z axis with no rotation, the vector spherical harmonics are related by:

$$\mathbf{M}^{(q)}_{nm,2} = \sum_{n'=0}^{\infty} A^{(m,q)}_{n,n'} \mathbf{M}^{(q)}_{n'm,1} + B^{(m,q)}_{n,n'} \mathbf{N}^{(q)}_{n'm,1}, \tag{19}$$

$$\mathbf{N}^{(q)}_{nm,2} = \sum_{n'=0}^{\infty} B^{(m,q)}_{n,n'} \mathbf{M}^{(q)}_{n'm,1} + A^{(m,q)}_{n,n'} \mathbf{N}^{(q)}_{n'm,1}, \tag{20}$$

where q denotes the order of the spherical Bessel functions ($q = 1, 2, 3, 4$). This relationship is valid in the region where $r > |d|$. The translation coefficients $A^{(m,q)}_{n,n'}$ and $B^{(m,q)}_{n,n'}$ are derived elsewhere. [19] We note that the expressions are:

$$\begin{aligned} A^{(m,q)}_{n,n'} &= C^{(m,q)}_{n,n'} - \frac{k_1 d}{n'+1}\sqrt{\frac{(n'-m+1)(n'+m+1)}{(2n'+1)(2n'+3)}}C^{(m,q)}_{n,n'+1} \\ &-\frac{k_1 d}{n'}\sqrt{\frac{(n'-m)(n'+m)}{(2n'+1)(2n'-1)}}C^{(m,q)}_{n,n'-1}, \end{aligned} \tag{21}$$

$$B^{(m,q)}_{n,n'} = \frac{-ik_1 m d}{n'(n'+1)} C^{(m,q)}_{n,n'}. \tag{22}$$

The $C^{(m,q)}_{n,n'}$ are scalar translation coefficients. Recursion expressions for these scalar coefficients can be derived using the method of Bobbert and Vlieger.[17] The details are shown elsewhere.[19] The necessary scalar translation coefficients are:

$$C^{(0,q)}_{0,n'} = \sqrt{2n'+1}\, j_{n'}(k_1 d), \tag{23}$$

$$C^{(0,q)}_{-1,n'} = -\sqrt{2n'+1}\, j_{n'}(k_1 d), \tag{24}$$

$$\begin{aligned} C^{(0,q)}_{n+1,n'} &= \frac{1}{(n+1)}\sqrt{\frac{2n+3}{2n'+1}}\left\{ n'\sqrt{\frac{2n+1}{2n'-1}}C^{(0,q)}_{n,n'-1} + \right. \\ &\left. n\sqrt{\frac{2n'+1}{2n-1}}C^{(0,q)}_{n-1,n'} - (n'+1)\sqrt{\frac{2n+1}{2n'+3}}C^{(0,q)}_{n,n'+1} \right\}, \end{aligned} \tag{25}$$

$$\sqrt{(n-m+1)(n+m)(2n'+1)}C_{n,n'}^{(m,q)} = \sqrt{(n'-m+1)(n'+m)(2n'+1)}C_{n,n'}^{(m-1,q)} - k_1 d\sqrt{\frac{(n'-m+2)(n'-m+1)}{(2n'+3)}}C_{n,n'+1}^{(m-1,q)} - k_1 d\sqrt{\frac{(n'+m)(n'+m-1)}{(2n'-1)}}C_{n,n'-1}^{(m-1,q)}, \qquad (26)$$

and.

$$C_{n,n'}^{(m,q)} = C_{n,n'}^{(-m,q)}. \qquad (27)$$

From these equations, we see that

$$A_{n,n'}^{(m,3)} = A_{n,n'}^{(m,4)} = A_{n,n'}^{(-m,3)} = A_{n,n'}^{(m)},$$
$$B_{n,n'}^{(m,3)} = B_{n,n'}^{(m,4)} = B_{n,n'}^{(-m,3)} = B_{n,n'}^{(m)}.$$
$$C_{n,n'}^{(m,3)} = C_{n,n'}^{(m,4)} = C_{n,n'}^{(-m,3)} = C_{n,n'}^{(m)}. \qquad (28)$$

The advantage of expanding the interior fields of the host sphere in the third and fourth kinds of spherical Bessel functions rather than the first and second kinds is that only one set of translation coefficients need to be calculated. Using equations 19, 20, and substituting those expressions into equation 12, we obtain

$$\begin{aligned}\mathbf{E}^2_{\text{ext}} = \sum_{n=0}^{\infty}\sum_{m=-n}^{n}\Bigg\{ & r_{nm}\left[\sum_{n'=0}^{\infty} A_{n,n'}^{(m,3)}\mathbf{M}_{n'm,1}^{(3)} + B_{n,n'}^{(m,3)}\mathbf{N}_{n'm,1}^{(3)}\right] \\ & + s_{nm}\left[\sum_{n'=0}^{\infty} B_{n,n'}^{(m,3)}\mathbf{M}_{n'm,1}^{(3)} + A_{n,n'}^{(m,3)}\mathbf{N}_{n'm,1}^{(3)}\right] \\ & + t_{nm}\left[\sum_{n'=0}^{\infty} A_{n,n'}^{(m,4)}\mathbf{M}_{n'm,1}^{(4)} + B_{n,n'}^{(m,4)}\mathbf{N}_{n'm,1}^{(4)}\right] \\ & + u_{nm}\left[\sum_{n'=0}^{\infty} B_{n,n'}^{(m,4)}\mathbf{M}_{n'm,1}^{(4)} + A_{n,n'}^{(m,4)}\mathbf{N}_{n'm,1}^{(4)}\right]\Bigg\}. \qquad (29)\end{aligned}$$

The field outside of the inclusion sphere is now expressed in terms of the vector spherical harmonics of the host sphere. Therefore, it is now possible to equate equation 29 with equation 5. Looking only at the $\mathbf{M}_{nm}^{(3)}$, we see that

$$\sum_{n=0}^{\infty}\sum_{m=-n}^{n} e_{nm}\mathbf{M}_{nm}^{(3)} = \sum_{n=0}^{\infty}\sum_{m=-n}^{n}\left[r_{nm}\sum_{n'=0}^{\infty} A_{n,n'}^{m} + s_{nm}\sum_{n'=0}^{\infty} B_{n,n'}^{m}\right]\mathbf{M}_{n'm}^{(3)}. \qquad (30)$$

Multiplying both sides of this equation by $\int M_{ij}^{(3)*} d^3r$ and using the orthogonality condition ($\delta_{ni}\delta_{mj}$ on the left hand side, and $\delta_{n'i}\delta_{mj}$ on the right hand side), we obtain

$$e_{ij} = \sum_{n=0}^{\infty} r_{nj} A_{n,i}^{j} + s_{nj} B_{ni}^{j}. \tag{31}$$

which can be rewritten as

$$e_{nm} = \sum_{n'=0}^{\infty} r_{n'm} A_{n',n}^{m} + s_{n'm} B_{n',n}^{m}. \tag{32}$$

Continuing the same line of reasoning, we can show

$$f_{nm} = \sum_{n'=0}^{\infty} s_{n'm} A_{n',n}^{m} + r_{n'm} B_{n',n}^{m}, \tag{33}$$

$$g_{nm} = \sum_{n'=0}^{\infty} t_{n'm} A_{n',n}^{m} + u_{n'm} B_{n',n}^{m}, \tag{34}$$

and

$$h_{nm} = \sum_{n'=0}^{\infty} u_{n'm} A_{n',n}^{m} + t_{n'm} B_{n',n}^{m}. \tag{35}$$

These are the four magic equations which relate the coefficients in the two different coordinate systems. Armed with these new expressions, it is now possible to replace one set of coefficients with another.

Substituting equations 32 and 34 into equation 6 we have

$$\begin{aligned} a_{nm} k_1 \psi_n(ka_1) + c_{nm} k_1 \xi_n^{(1)}(ka_1) &= k\xi_n^{(1)}(k_1 a_1) \sum_{n'=0}^{\infty} r_{n'm} A_{n,n'}^{m} + s_{n'm} B_{n,n'}^{m} + \\ & \quad k\xi_n^{(2)}(k_1 a_1) \sum_{n'=0}^{\infty} t_{n'm} A_{n,n'}^{m} + u_{n'm} B_{n,n'}^{m}. \end{aligned} \tag{36}$$

Using equations 17 and 18 for $r_{n'm}$ and $s_{n'm}$, we can substitute those expressions into the above equation to obtain

$$\begin{aligned} a_{nm} \psi_n(ka_1) + c_{nm} \xi_n^{(1)}(ka_1) &= \frac{k}{k_1} \sum_{n'=0}^{\infty} t_{n'm} A_{n,n'}^{m} \left[\xi_n^{(2)}(k_1 a_1) + Q_{n'}^{r} \xi_n^{(1)}(k_1 a_1)\right] + \\ & \quad u_{n'm} B_{n,n'}^{m} \left[\xi_n^{(2)}(k_1 a_1) + Q_{n'}^{s} \xi_n^{(1)}(k_1 a_1)\right]. \end{aligned} \tag{37}$$

Likewise, we can substitute equations 32 - 35 into equations 7 - 9 (and together with equations 17 and 18) to obtain the following set of simultaneous equations, which can be solved to yield the scattering coefficients (c_{nm} and d_{nm}) or the internal field coefficients (t_{nm} and u_{nm}) of the host sphere:

$$a_{nm}\psi_n'(ka_1) + c_{nm}\xi_n'^{(1)}(ka_1) = \sum_{n'=0}^{\infty} t_{n'm}A^m_{n,n'}\left[\xi_n'^{(2)}(k_1a_1) + Q^r_{n'}\xi_n'^{(1)}(k_1a_1)\right] + u_{n'm}B^m_{n,n'}\left[\xi_n'^{(2)}(k_1a_1) + Q^s_{n'}\xi_n'^{(1)}(k_1a_1)\right], \quad (38)$$

$$b_{nm}\psi_n(ka_1) + d_{nm}\xi_n^{(1)}(ka_1) = \sum_{n'=0}^{\infty} t_{n'm}B^m_{n,n'}\left[\xi_n^{(2)}(k_1a_1) + Q^r_{n'}\xi_n^{(1)}(k_1a_1)\right] + u_{n'm}A^m_{n,n'}\left[\xi_n^{(2)}(k_1a_1) + Q^s_{n'}\xi_n^{(1)}(k_1a_1)\right]. \quad (39)$$

$$b_{nm}\psi_n'(ka_1) + d_{nm}\xi_n'^{(1)}(ka_1) = \frac{k}{k_1}\sum_{n'=0}^{\infty} t_{n'm}B^m_{n,n'}\left[\xi_n'^{(2)}(k_1a_1) + Q^r_{n'}\xi_n'^{(1)}(k_1a_1)\right] + u_{n'm}A^m_{n,n'}\left[\xi_n'^{(2)}(k_1a_1) + Q^s_{n'}\xi_n'^{(1)}(k_1a_1)\right]. \quad (40)$$

2.4 Scattering Coefficients

The set of four equations (37 - 40) is the final product in our search for the scattering coefficients. We have four equations and four unknowns, the unknowns being c_{nm}, d_{nm}, t_{nm}, and u_{nm}. The interior field coefficients inside the host sphere, t_{nm} and u_{nm}, (which can also be viewed as the exterior field coefficients of the inner sphere) may be calculated by eliminating the scattering coefficients c_{nm} and d_{nm} in equations 37 - 40. Since all the elements comprising the matrix in the solution are known except for the coefficients t_{nm} and u_{nm}, we can perform an LU-decomposition to solve for those interior field coefficients. The matrix representation is:

$$a_{nm}\gamma_n = \sum_{n'=0}^{\infty} t_{n'm}T^{(m,1)}_{n,n'} + u_{n'm}U^{(m,1)}_{n,n'}, \quad (41)$$

$$b_{nm}\gamma_n = \sum_{n'=0}^{\infty} t_{n'm}T^{(m,2)}_{n,n'} + u_{n'm}U^{(m,2)}_{n,n'}. \quad (42)$$

where

$$\gamma_n = k_1\left[\xi_n'^{(1)}(ka_1)\psi_n(ka_1) - \psi_n'(ka_1)\xi_n^{(1)}(ka_1)\right], \quad (43)$$

$$T^{(m,1)}_{n,n'} = A^m_{n,n'}\left\{k\xi_n'^{(1)}(ka_1)\left[\xi_n^{(2)}(k_1a_1) + Q^r_{n'}\xi_n^{(1)}(k_1a_1)\right] - k_1\xi_n^{(1)}(ka_1)\left[\xi_n'^{(2)}(k_1a_1) + Q^r_{n'}\xi_n'^{(1)}(k_1a_1)\right]\right\}, \quad (44)$$

$$T_{n,n'}^{(m,2)} = B_{n,n'}^{m}\left\{k_1\xi_n'^{(1)}(ka_1)\left[\xi_n^{(2)}(k_1a_1)+Q_{n'}^{r}\xi_n^{(1)}(k_1a_1)\right]\right.$$
$$\left.-k\xi_n^{(1)}(ka_1)\left[\xi_n'^{(2)}(k_1a_1)+Q_{n'}^{r}\xi_n'^{(1)}(k_1a_1)\right]\right\}, \quad (45)$$

$$U_{n,n'}^{(m,1)} = B_{n,n'}^{m}\left\{k\xi_n'^{(1)}(ka_1)\left[\xi_n^{(2)}(k_1a_1)+Q_{n'}^{s}\xi_n^{(1)}(k_1a_1)\right]\right.$$
$$\left.-k_1\xi_n^{(1)}(ka_1)\left[\xi_n'^{(2)}(k_1a_1)+Q_{n'}^{s}\xi_n'^{(1)}(k_1a_1)\right]\right\}. \quad (46)$$

$$U_{n,n'}^{(m,2)} = A_{n,n'}^{m}\left\{k_1\xi_n'^{(1)}(ka_1)\left[\xi_n^{(2)}(k_1a_1)+Q_{n'}^{s}\xi_n^{(1)}(k_1a_1)\right]\right.$$
$$\left.-k\xi_n^{(1)}(ka_1)\left[\xi_n'^{(2)}(k_1a_1)+Q_{n'}^{s}\xi_n'^{(1)}(k_1a_1)\right]\right\}. \quad (47)$$

The scattering coefficients, c_{nm} and d_{nm} may then be calculated using equations 37 - 40. Once we know the scattering coefficients, we can determine the scattering matrix elements and other parameters associated with the scatter.

2.5 Plane Wave Expansion

Since the small sphere is centered on the z axis, for the solution to be completely general, the incident plane wave must be in an arbitrary direction in the x-z plane. Two plane waves must be considered in order to account for each polarization state. When the plane wave is polarized perpendicular to the x-z plane (TE), the coefficients are found to be [18]:

$$a_{nm} = a_{nm}^{TE} = \frac{i^n}{n(n+1)}\left[\sqrt{(n-m)(n+m+1)}\tilde{P}_n^{m+1}(\cos\alpha)\right.$$
$$\left.-\sqrt{(n-m+1)(n+m)}\tilde{P}_n^{m-1}(\cos\alpha)\right]. \quad (48)$$

$$= \frac{2i^{n+2}}{n(n+1)}\frac{\partial}{\partial\alpha}\tilde{P}_n^m(\cos\alpha). \quad (49)$$

$$b_{nm} = b_{nm}^{TE} = \frac{i^{n+2}(2n+1)}{n(n+1)}\left[\sqrt{\frac{(n-m+1)(n-m+2)}{(2n+1)(2n+3)}}\tilde{P}_{n+1}^{m-1}(\cos\alpha)\right.$$
$$\left.+\sqrt{\frac{(n+m+1)(n+m+2)}{(2n+1)(2n+3)}}\tilde{P}_{n+1}^{m+1}(\cos\alpha)\right], \quad (50)$$

$$= \frac{2i^{n+2}}{n(n+1)}\frac{m\tilde{P}_n^m(\cos\alpha)}{\sin\alpha}. \quad (51)$$

When the plane wave is polarized in the x-z plane (TM), the coefficients are found to be

$$a_{nm} = a_{nm}^{TM} = ib_{nm}^{TE}. \tag{52}$$

and

$$b_{nm} = b_{nm}^{TM} = ia_{nm}^{TE}. \tag{53}$$

2.6 The Scattering Amplitudes and Efficiencies

We consider the scattering amplitudes in the far field, where $kr_1 \gg ka$. The scattered fields in this case are in the $\hat{\theta}$ and $\hat{\varphi}$ directions. In this limit, the spherical Hankel function reduces to spherical waves:

$$h_n^{(1)}(kr) \sim \frac{(-i)^n}{ikr} e^{ikr}. \tag{54}$$

The scattering amplitudes can be expressed in the form of the matrix,

$$\begin{pmatrix} E_{\varphi}^{sca} \\ E_{\theta}^{sca} \end{pmatrix} = \frac{e^{ikr_1}}{-ikr_1} \begin{pmatrix} S_1 & S_4 \\ S_3 & S_2 \end{pmatrix} \begin{pmatrix} E_{TE}^{inc} \\ E_{TM}^{inc} \end{pmatrix}. \tag{55}$$

The scattering amplitude matrix elements are solved by expanding the scattered electric fields (equation 4) in terms of the vector wave functions and then expanding the vector wave functions (equation 1) in terms of the polarization directions. Since we are concerned with the far field, we may drop the $\hat{r}$ component and keep only the $\hat{\theta}$ and $\hat{\varphi}$ components. After some algebra, we have:

$$S_1 = \sum_{n=0}^{\infty} \sum_{m=-n}^{n} (-i)^n e^{im\varphi_1} \times \left[d_{nm}^{TE} \frac{m}{\sin\theta_1} \tilde{P}_n^m(\cos\theta_1) + c_{nm}^{TE} \frac{\partial}{\partial\theta_1} \tilde{P}_n^m(\cos\theta_1) \right]. \tag{56}$$

$$S_2 = -i \sum_{n=0}^{\infty} \sum_{m=-n}^{n} (-i)^n e^{im\varphi_1} \times \left[c_{nm}^{TM} \frac{m}{\sin\theta_1} \tilde{P}_n^m(\cos\theta_1) + d_{nm}^{TM} \frac{\partial}{\partial\theta_1} \tilde{P}_n^m(\cos\theta_1) \right]. \tag{57}$$

$$S_3 = -i \sum_{n=0}^{\infty} \sum_{m=-n}^{n} (-i)^n e^{im\varphi_1} \times \left[c_{nm}^{TE} \frac{m}{\sin\theta_1} \tilde{P}_n^m(\cos\theta_1) + d_{nm}^{TE} \frac{\partial}{\partial\theta_1} \tilde{P}_n^m(\cos\theta_1) \right]. \tag{58}$$

$$S_4 = \sum_{n=0}^{\infty} \sum_{m=-n}^{n} (-i)^n e^{im\varphi_1} \times \left[d_{nm}^{TM} \frac{m}{\sin\theta_1} \tilde{P}_n^m(\cos\theta_1) + c_{nm}^{TM} \frac{\partial}{\partial\theta_1} \tilde{P}_n^m(\cos\theta_1) \right]. \tag{59}$$

Using the following relationship for normalized, associated Legendre polynomials,

$$\tilde{P}_n^{-m}(\cos\theta_1) = (-1)^m \tilde{P}_n^m(\cos\theta_1), \tag{60}$$

the following relationships between the scattering coefficients may be derived:

$$c_{n\bar{m}}^{TE} = (-1)^m c_{nm}^{TE},$$

$$
\begin{aligned}
{}^{TM}_{n\bar{m}} &= (-1)^{m+1} c^{TM}_{nm}, \\
d^{TE}_{n\bar{m}} &= (-1)^{m+1} d^{TE}_{nm}; \\
d^{TM}_{n\bar{m}} &= (-1)^{m} d^{TM}_{nm}.
\end{aligned} \tag{61}
$$

where $\bar{m} = -m$.

The scattering, extinction, and absorption efficiencies of the system are defined as the cross sections per projected area and may be expressed as

$$
Q_{sca} = \frac{2}{(ka_1)^2} \times \left[\sum_{n=1}^{\infty} n(n+1) \sum_{m=-n}^{n} \left(\left| c^{TE}_{nm} \right|^2 + \left| d^{TE}_{nm} \right|^2 + \left| c^{TM}_{nm} \right|^2 + \left| d^{TM}_{nm} \right|^2 \right) \right], \tag{62}
$$

$$
Q_{ext} = \frac{-2}{(ka_1)^2} \times Re \left[\sum_{n=1}^{\infty} n(n+1) \sum_{m=-n}^{n} \left(c^{TE}_{nm} a^{TE*}_{nm} + d^{TE}_{nm} b^{TE*}_{nm} + c^{TM}_{nm} a^{TM*}_{nm} + d^{TM}_{nm} b^{TM*}_{nm} \right) \right]. \tag{63}
$$

$$
Q_{abs} = Q_{ext} - Q_{sca}. \tag{64}
$$

where a^*_{nm} and b^*_{nm} are the complex conjugates of a_{nm} and b_{nm}, respectively. The asymmetry parameter, g, can be expressed as

$$
\begin{aligned}
g \;=\; & \frac{4}{Q_{sca}(ka)^2} \sum_{n,m} m Re \left(c^{TE}_{nm} d^{TE*}_{nm} + c^{TM}_{nm} d^{TM*}_{nm} \right) \\
& + n(n+2) \sqrt{\frac{(n-m+1)(n+m+1)}{(2n+3)(2n+1)}} \times \\
& Re \left[i(c^{TE}_{nm} c^{TE*}_{n+1m} + d^{TE}_{nm} d^{TE*}_{n+1m} + c^{TM}_{nm} c^{TM*}_{n+1m} + d^{TM}_{nm} d^{TM*}_{n+1m} \right].
\end{aligned} \tag{65}
$$

Detailed derivations for the asymmetry parameter and the efficiencies are given elsewhere.[19]

3 SPECIAL CASES

Under certain conditions, the equations describing the internal and scattered fields may be simplified. In this section we examine some of these cases.

3.1 Simplifications of Q^r_n and Q^s_n

When the inclusion satisfies certain conditions, equations 17 and 18, which describe the coefficients Q^r_n and Q^s_n, can be simplified. In this subsection we demonstrate these simplifications. First, when the optical size of the inclusion is much smaller than the wavelength ($k_2 a_2 \ll 1$).

$$
Q^r_n \sim 1, \tag{66}
$$

$$Q_n^s \sim 1 + \frac{4i(k_2a_2)^3}{3}\frac{(m_2^2 - m_1)}{(m_2^2 + 2m_1)}. \tag{67}$$

These are similar to the scattering coefficients for a Rayleigh scatterer. If the inclusion is a perfectly conducting medium ($m_2 \rightarrow \infty$), the coefficients can be reduced to

$$Q_n^r = -\frac{\xi_n^{(2)}(k_1a_2)}{\xi_n^{(1)}(k_1a_2)}, \tag{68}$$

$$Q_n^s = -\frac{\xi_n'^{(2)}(k_1a_2)}{\xi_n'^{(1)}(k_1a_2)}. \tag{69}$$

When the perfectly conducting inclusion is much smaller than the wavelength ($a_2 \ll \lambda$), equations 68 and 69 further reduce to

$$Q_n^r \sim 1 + \frac{2i(k_1a_2)^3}{3}; \tag{70}$$

$$Q_n^s \sim 1 - \frac{4i(k_1a_2)^3}{3}. \tag{71}$$

3.2 Mie Reduction

It should be noted that if the refractive index of the inner sphere is equal to that of the outer sphere, then the exterior field coefficients of the inner sphere, Q_n^r and Q_n^s are unity (equations 17 and 18). The Riccati-Hankel functions of the first and second kind on the right hand sides of equations 37 - 40 combine to form Riccati-Bessel functions of the first kind:

$$a_{nm}\psi_n(ka_1) + c_{nm}\xi_n^{(1)}(ka_1) = \frac{2k}{k_1}\sum_{n'=0}^{\infty} t_{n'm}A^m_{n,n'}\psi_n(k_1a_1) + u_{n'm}B^m_{n,n'}\psi_n(k_1a_1). \tag{72}$$

$$a_{nm}\psi_n'(ka_1) + c_{nm}\xi_n'^{(1)}(ka_1) = 2\sum_{n'=0}^{\infty} t_{n'm}A^m_{n,n'}\psi_n'(k_1a_1) + u_{n'm}B^m_{n,n'}\psi_n'(k_1a_1). \tag{73}$$

$$b_{nm}\psi_n(ka_1) + d_{nm}\xi_n^{(1)}(ka_1) = 2\sum_{n'=0}^{\infty} t_{n'm}B^m_{n,n'}\psi_n(k_1a_1) + u_{n'm}A^m_{n,n'}\psi_n(k_1a_1), \tag{74}$$

$$b_{nm}\psi_n'(ka_1) + d_{nm}\xi_n'^{(1)}(ka_1) = \frac{2k}{k_1}\sum_{n'=0}^{\infty} t_{n'm}B^m_{n,n'}\psi_n'(k_1a_1) + u_{n'm}A^m_{n,n'}\psi_n'(k_1a_1). \tag{75}$$

Physically, the Riccati-Hankel functions of the first and second kind represent outgoing and incoming spherical waves, respectively. With the inner sphere removed, there is no scattering object within the larger sphere, and standing waves represented by Riccati-Bessel functions of the first kind are contained within its interior. Note that substituting equations 34 and 35 into the right hand side of equations 72 - 75 yields the boundary equations for Mie theory.

3.3 Concentric Spheres

When the host and inclusion spheres are concentric, i.e. they share the same origin ($d = 0$), the translation coefficients are simplified, and

$$A^m_{n,n'} = \delta_{nn'},$$
$$B^m_{n,n'} = \delta_{nn'}; \tag{76}$$

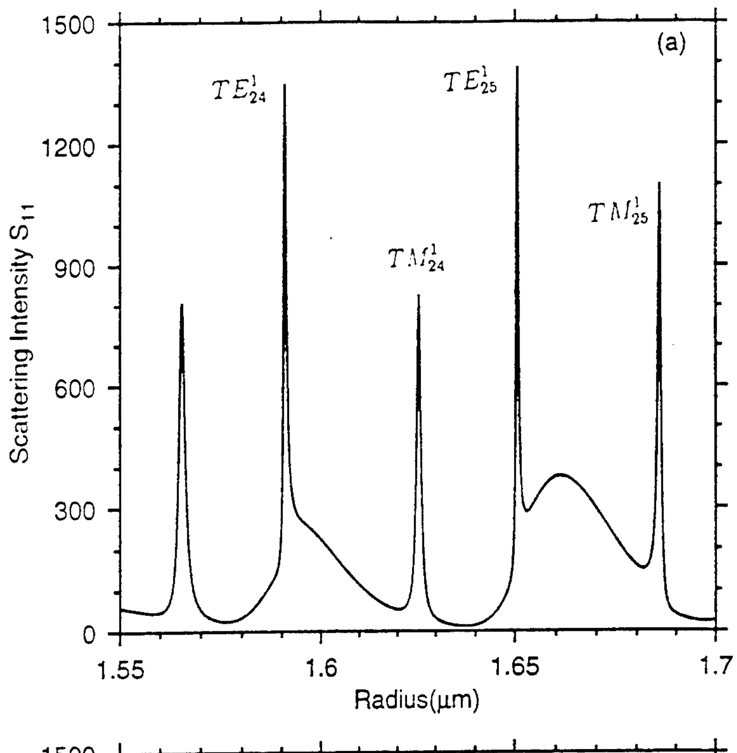
where $\delta_{nn'}$ is the Kronecker delta function. The scattering coefficients described by equations 37 - 40 now reduce to the accepted concentric spheres expressions given by Aden and Kerker.[4]

4 RESULTS AND DISCUSSIONS

In this section we examine the scattering of the nonconcentric sphere system as a function of system parameters. As in any new theory, confidence of a new result relies on its simplification to previously known solutions. But rather than analytically simplifying the general nonconcentric expressions into specialized cases (we have already shown the special cases in the previous section), we will keep everything general and allow the computer to numerically give us the desired results. The greatest advantage of doing it this way is the assurance that the program is coded correctly; or at the very least, the code reduces correctly in the specialized cases. Simplifications of the system can be made by assigning to it special values so that comparisons with Mie theory and concentric spheres theory may be done. We will show that when we simplify the system, we obtain numerical values that are identical to Mie theory and concentric spheres theory. We then consider two specific but arbitrary cases of the noncencentric system. The first is an air inclusion ($\tilde{m}_2 = 1.0 + 0i$) in a host sphere composed of water ($\tilde{m}_1 = 1.33 + 0i$). The second is a perfectly conducting inclusion in a host sphere composed of water. In both cases the host radius is equal to the wavelength of the incoming radiation ($a_1 = \lambda$).

4.1 Comparison with Mie Theory

When the refractive index of the host ($\tilde{m}_1$) and the refractive index of the inclusion ($\tilde{m}_2$) are identical, we can say that the system is homogeneous. In this case, the nonconcentric sphere should reduce to Mie theory. We take the host and the inclusion to be glycerol spheres with refractive index $\tilde{m} = 1.4746 + 0i$ at wavelength $\lambda = 0.5145\mu m$. Figure 3a shows the scattering intensity of the glycerol sphere as a function of the sphere's radius. This plot is obtained using the Mie code presented by Bohren and Huffman.[20] The familiar quasi-periodic resonance peaks are readily observable. Figure 3b shows the same plot, but using the nonconcentric sphere model. We note that all the cycles of resonances are the same; in fact, we have normalized the scattering intensity so that numeric values from the Mie code and the nonconcentric code are the same (as can be seen in Figure 3). Also listed in Table 1 are the results from a run using the



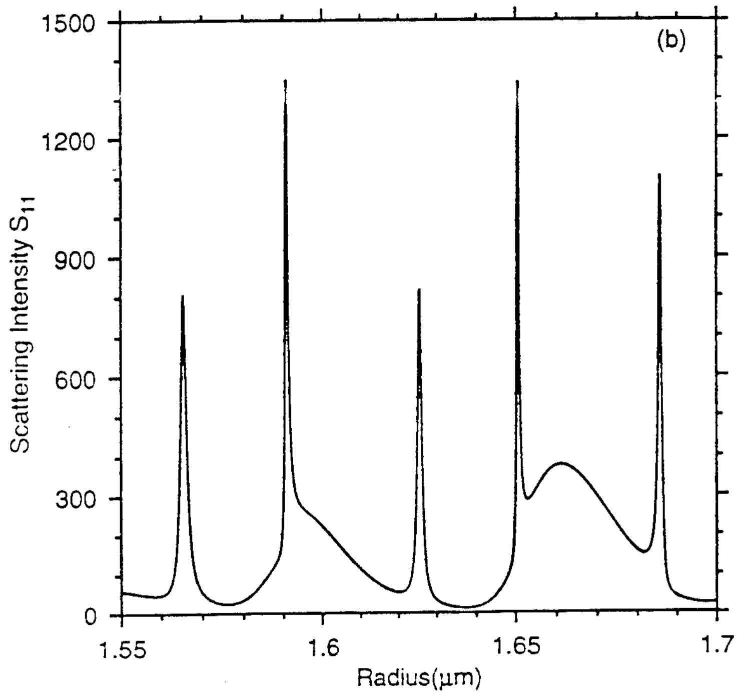


Figure 3. Comparison of the scattering intensity as a function of droplet radius ($\tilde{m} = 1.4746 + 0i$) using the Mie code from Bohren and Huffman (a), and the nonconcentric code (b) with $\tilde{m}_1 = \tilde{m}_2 = 1.4746 + 0i$. The resonance structures are clearly evident in both cases.

TABLE 1

Comparison with Mie Scattering output

Input:

REFRE = 1.550 REFIM = 0.0

SPHERE RADIUS = 0.525 μm

WAVELENGTH = 0.6328 μm

Output:

Qsca = 3.10543 Qext = 3.10543 g = 0.633137

angle	S11	POL (S12)	S33	S34
0.0	1.00000	0.0	1.00000	0.0
9.0	0.785391	4.59811E-03	0.999400	3.43261E-02
18.0	0.356897	4.58540E-02	0.986022	0.160184
27.0	7.66119E-02	0.364744	0.843602	0.394077
36.0	3.55355E-02	0.534997	0.686967	-0.491787
45.0	7.01845E-02	-9.59953E-03	0.959825	-0.280434
54.0	5.74324E-02	-4.77927E-02	0.985371	0.163584
63.0	2.19660E-02	0.440604	0.648043	0.621216
72.0	1.25959E-02	0.831996	0.203255	-0.516208
81.0	1.73750E-02	-3.41670E-02	0.795354	-0.605182
90.0	1.24601E-02	-0.230462	0.937497	0.260742
99.0	6.79093E-03	0.713472	-7.17406E-03	0.700647
108.0	9.54240E-03	0.756255	-3.94746E-02	-0.653085
117.0	8.63419E-03	0.281215	0.536251	-0.795835
126.0	2.27421E-03	0.239612	0.967602	7.95797E-02
135.0	5.43998E-03	0.850803	0.187531	-0.490882
144.0	1.60244E-02	0.706334	0.495255	-0.505781
153.0	1.88853E-02	0.891081	0.453277	-2.26817E-02
162.0	1.95254E-02	0.783320	-0.391613	0.482752
171.0	3.01676E-02	0.196194	-0.962069	0.189556
180.0	3.83189E-02	0.0	-1.00000	0.0

TABLE 2
Comparison of Nonconcentric with Coated Sphere Theory

Input:

$\lambda = 3.0\mu m$, $m_1 = 1.409 + 0.1747i$, $m_2 = 1.59 + 0.66i$

Output:

Radius (μm)	Coated Sphere	Nonconcentric
$a_1 = 6.2650$	$Q_{ext} = 2.32803$	$Q_{ext} = 2.32803$
$a_2 = 0.1710$	$Q_{sca} = 1.14341$	$Q_{sca} = 1.143413$
$a_1 = 5.50$	$Q_{ext} = 2.34481$	$Q_{ext} = 2.344814$
$a_2 = 1.50$	$Q_{sca} = 1.13227$	$Q_{sca} = 1.132269$
$a_1 = 3.50$	$Q_{ext} = 2.58325$	$Q_{ext} = 2.58325$
$a_2 = 2.00$	$Q_{sca} = 1.28485$	$Q_{sca} = 1.28485$

Input:

$\lambda = 3.0\mu m$, $m_1 = 1.409 + 0.0i$, $m_2 = 1.592 + 0.66i$

Output:

Radius (μm)	Coated Sphere	Nonconcentric
$a_1 = 6.2650$	$Q_{ext} = 2.77588$	$Q_{ext} = 2.77588$
$a_2 = 0.1710$	$Q_{sca} = 2.77518$	$Q_{sca} = 1.77518$
$a_1 = 5.50$	$Q_{ext} = 2.04387$	$Q_{ext} = 2.04387$
$a_2 = 1.50$	$Q_{sca} = 1.85917$	$Q_{sca} = 1.859173$
$a_1 = 3.50$	$Q_{ext} = 3.19478$	$Q_{ext} = 3.194786$
$a_2 = 2.00$	$Q_{sca} = 2.41198$	$Q_{sca} = 2.411986$

Input:

$\lambda = 3.0\mu m$, $m_1 = 1.409 + 0.0i$, $m_2 = 1.592 + 0.0i$

Output:

Radius (μm)	Coated Sphere	Nonconcentric
$a_1 = 6.2650$	$Q_{ext} = 2.77604$	$Q_{ext} = 2.77604$
$a_2 = 0.1710$	$Q_{sca} = 2.77604$	$Q_{sca} = 2.77604$
$a_1 = 5.50$	$Q_{ext} = 2.31972$	$Q_{ext} = 2.319716$
$a_2 = 1.50$	$Q_{sca} = 2.31972$	$Q_{sca} = 2.319716$
$a_1 = 3.50$	$Q_{ext} = 2.28173$	$Q_{ext} = 2.28173$
$a_2 = 2.00$	$Q_{sca} = 2.28173$	$Q_{sca} = 2.28173$

nonconcentric code with inputs taken from page 482 of Bohren and Huffman.[20] The numerical values for both programs are identical. This tells us that our nonconcentric sphere model does indeed reduce to Mie theory in the appropriate limit.

4.2 Comparison with Concentric Theory

When the displacement d separating the two origins is zero, we have a concentric sphere system. Using the concentric code presented in Bohren and Huffman, we are able to show that, in all cases, our nonconcentric sphere model reduces to the concentric sphere model. Sample results of using the two methods are given in Table 2 (the first set of output can be compared to page 489 of Bohren and Huffman).[20]

4.3 Examination of Two Types of Inclusion

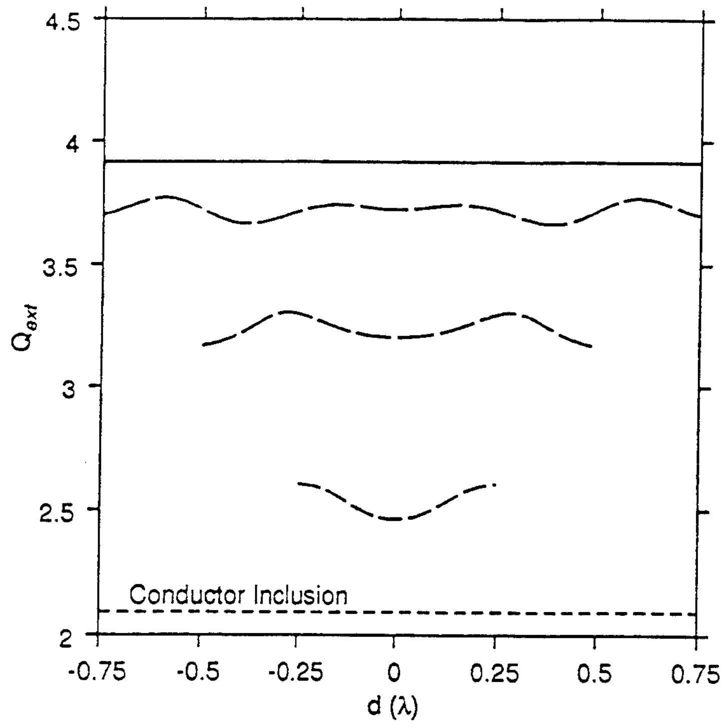
In this section, we examine the scattering properties of the nonconcentric sphere model by considering two specific but arbitrary cases. The first is an air inclusion ($\tilde{m}_2 = 1.0 + 0.0i$) in a host sphere composed of water ($\tilde{m}_1 = 1.33 + 0.0i$). The second is a perfectly conducting inclusion in a host sphere composed of water. In both of these cases, the host radius is equal to the wavelength of the incident radiation ($a_1 = \lambda$).

Figure 4 shows the extinction efficiencies as a function of displacement distance d when the incident angle $\alpha = 0^0$ and the beam travels parallel to the z axis for inclusion spheres of different radii. For comparison, we also plot the extinction efficiencies for a sphere composed entirely of water ($a_2 = 0$, shown by the solid lines, top and bottom), for which $Q_{ext} = 3.916$. For a sphere composed entirely of a perfectly conducting medium (top graph), $Q_{ext} = 2.094$. For a sphere composed entirely of air, we have, of course, $Q_{ext} = 0.0$. The most remarkable feature of the graphs is their symmetry. The extinction is independent of the sign of the displacement; i.e. $Q_{ext}(d) = Q_{ext}(-d)$.

The result that $Q_{ext}(d) = Q_{ext}(-d)$ is a direct consequence of the reciprocity theorem which states that if a scattering system whose scattering matrix is given by equation 55 is replaced by its mirror image (the mirror is perpendicular to the direction of the incident field), then the scattering due to the mirror system in the forward direction ($\theta_1 = \alpha$, $\varphi_1 = 0^0$) is given by

$$\begin{pmatrix} E_{\varphi}^{sca}(\alpha, 0^0) \\ E_{\theta}^{sca}(\alpha, 0^0) \end{pmatrix} = \frac{e^{ikr_1}}{-ikr_1} \begin{pmatrix} S_1 & -S_3 \\ -S_4 & S_2 \end{pmatrix} \begin{pmatrix} E_{TE}^{inc} \\ E_{TM}^{inc} \end{pmatrix}. \tag{77}$$

We know from the optical theorem that the extinction is proportional to the real part of the scattering amplitude in the forward direction, and from equations 48-53 and 56-61, it can be shown that there is no coupling of modes in the scattering plane, i.e., S_3 and S_4 do not contribute to the scattering amplitude when $\varphi = 0^0$. Therefore, the extinction of the scattering system discussed in this paper is equal to the extinction of its mirror image.



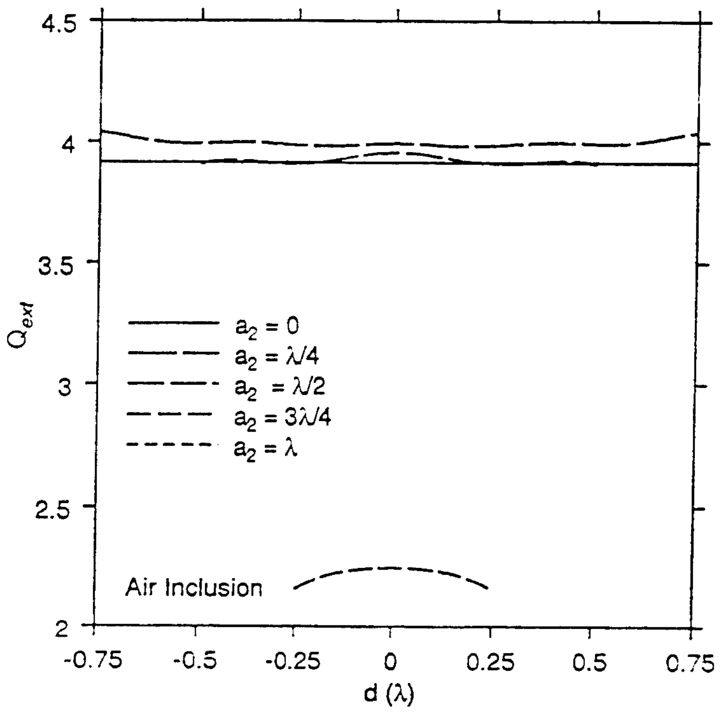
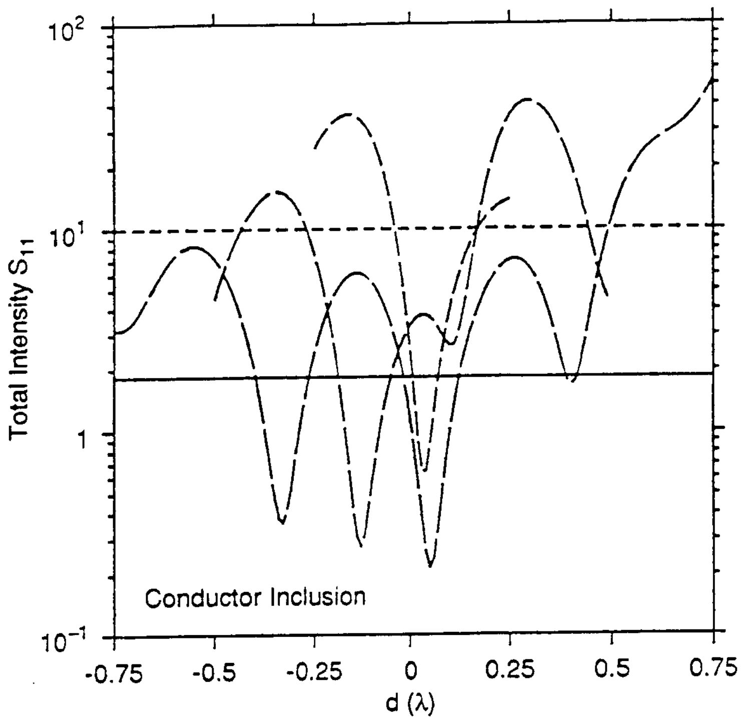


Figure 4. The extinction efficiencies as a function of displacement distance d when the incident angle $\alpha = 0°$ for a perfectly conducting inclusion (top) and an air inclusion (bottom) inside an $a_1 = \lambda$ water host ($\tilde{m}_1 = 1.33 + 0i$).



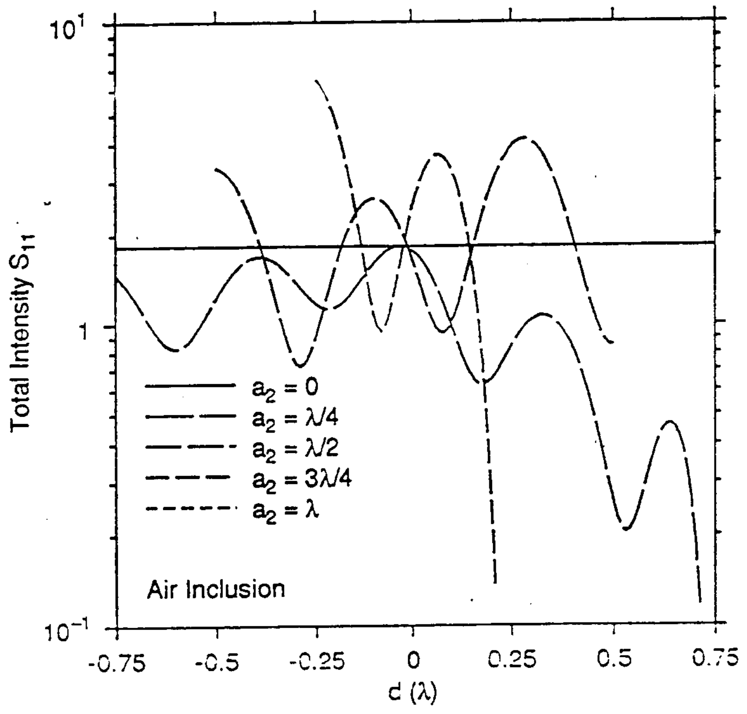


Figure 5. The backscattering total intensity S_{11} as a function of displacement distance d when the incident angle $\alpha = 0°$ for a perfectly conducting inclusion (top) and an air inclusion (bottom) inside an $a_1 = \lambda$ water host ($\tilde{m}_1 = 1.33 + 0i$).

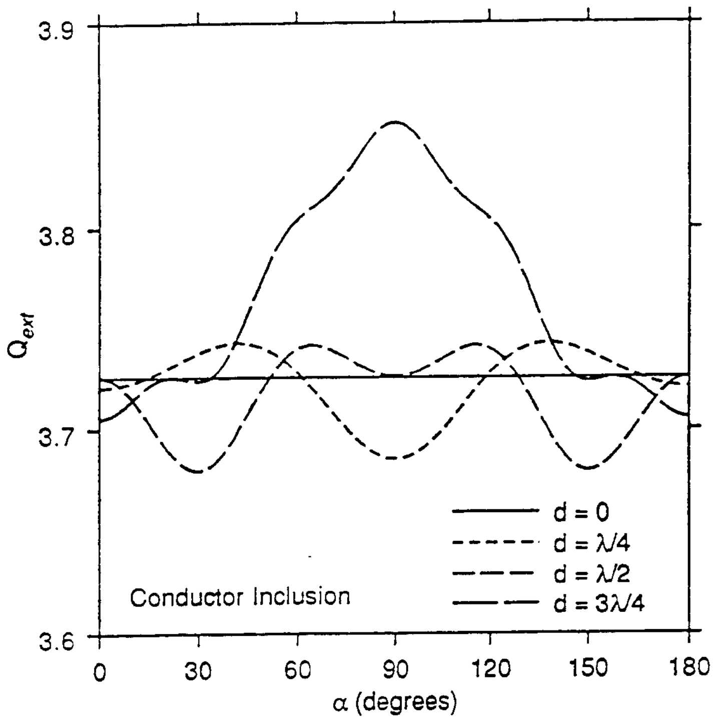
Figure 5 shows the backscattering intensities (incident angle $\alpha = 0^0$ and $\theta_1 = 180^0$) as a function of displacement distance d for the same cases shown in Figure 4. Also shown on both graphs are the backscattering intensities when there is no inclusion ($a_2 = 0$), in which case the backscattering $S_{11} = 1.837$. Unlike the plots shown in Figure 4, these plots do not show a symmetric S_{11}. There are also some differences in the backscattered intensities between the two systems in this figure. For the case of the perfectly conducting inclusion, the backscatter is generally greater than for a sphere with no inclusion (at times by more than an order of magnitude); whereas, for the case of the air inclusion, the backscatter generally stays within a factor of two of the backscatter of the system with no inclusion.

The difference in the backscattering between a system with a perfectly conducting inclusion and an air inclusion can be better understood by considering the rays which strike the system. Incident rays striking the surface of the host sphere either refract convergently within the sphere (for our case when the refractive index of the host sphere is greater than the refractive index of the incident medium) or reflect off the outer surface. Some of the refracted rays which reflect off the surface of the inclusion may be traced into the backscatter direction (these are the rays which strike the inclusion at normal incident or strike the inclusion at $x_2 = y_2 = 0$). The reflectance from the perfectly conducting inclusion is 100%; whereas, the reflectance from an air-water interface is approximately 2% near normal incidence. Because of the greater reflectance at the conductor interface, we expect the backscatter to be greater when there is a perfectly conducting inclusion. The reflectance at the air inclusion interface is approximately the same as at the outer surface interface, so we do not expect the backscatter for the air-inclusion system to vary as greatly from the sphere system without the inclusion.

Figure 6 shows the extinction efficiencies for an $a_2 = \lambda/4$ inclusion as a function of the incident angle α for four displacements. When the inclusion and host spheres are concentric, the extinction does not vary, as expected from symmetry. As the inclusion distance increases, the magnitude of the total variation in the extinction becomes greater. Also, the number of inflection points in the extinction curves increases. Essentially, the closer the inclusion is to the host sphere surface, the greater the effect it has on the extinction of the system. This point is reinforced in the next section where we examine the resonance of the composite system. We also note that because of the system symmetry, the extinction efficiencies must be symmetric about $\alpha = 0^0$, and from the reciprocity theorem, they must also be symmetric about $\alpha = 90^0$.

Figure 7 shows the backscattering intensities ($\theta_1 = \pi + \alpha$) for an $a_2 = \lambda/4$ inclusion as a function of incident angle α for the same displacements as in the previous figure. As was the case for the extinction efficiencies, the number of inflection points in these backscattering intensity curves increases as the inclusion distance increases. However, the symmetry about $\alpha = 90^0$ which exists in the extinction efficiencies of Figure 6 is not present in the backscattering intensities.

Figure 8 shows the extinction efficiencies as a function of the inclusion radius for four



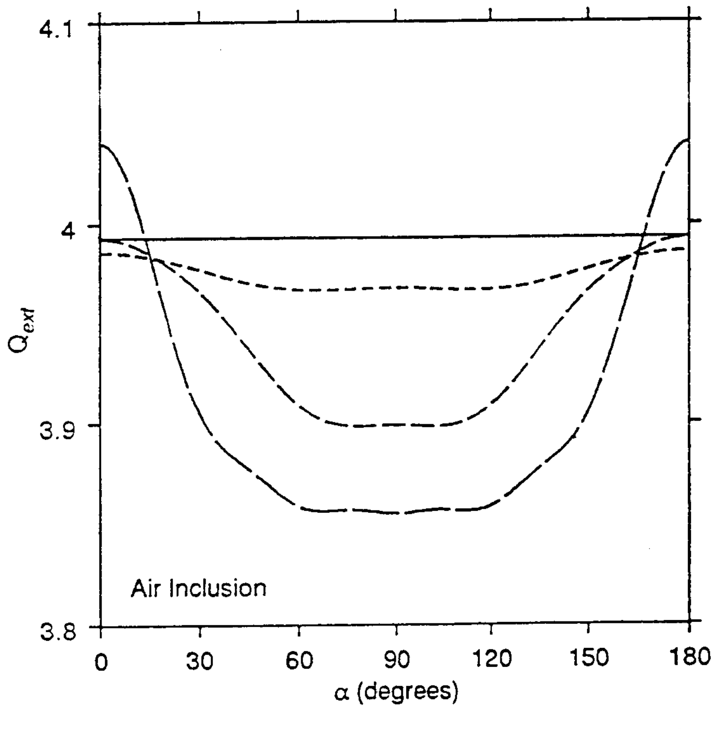
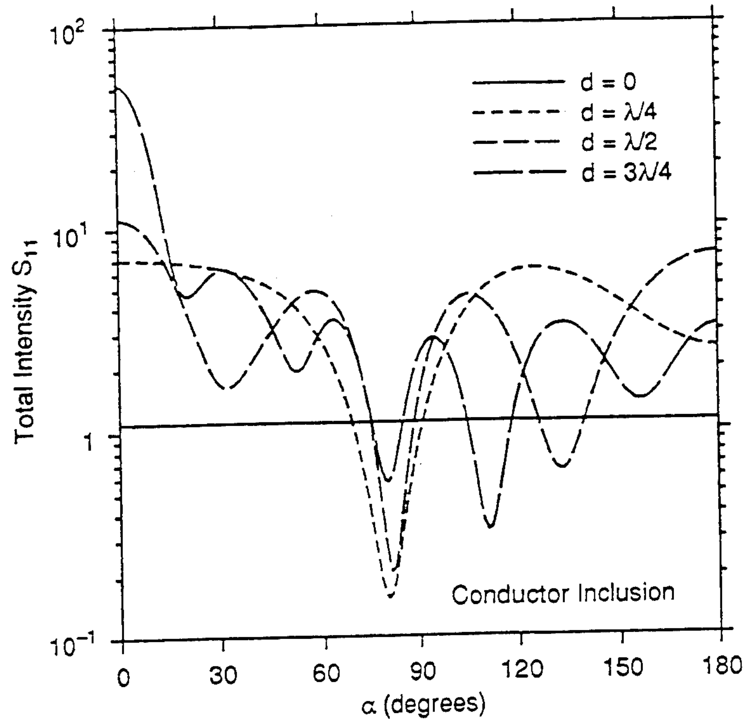


Figure 6. The extinction efficiencies as a function of the incident angle α for a perfectly conducting inclusion (top) and an air inclusion (bottom) inside an $a_1 = \lambda$ water host ($\tilde{m}_1 = 1.33 + 0i$). The inclusion radius is $a_2 = \lambda/4$.



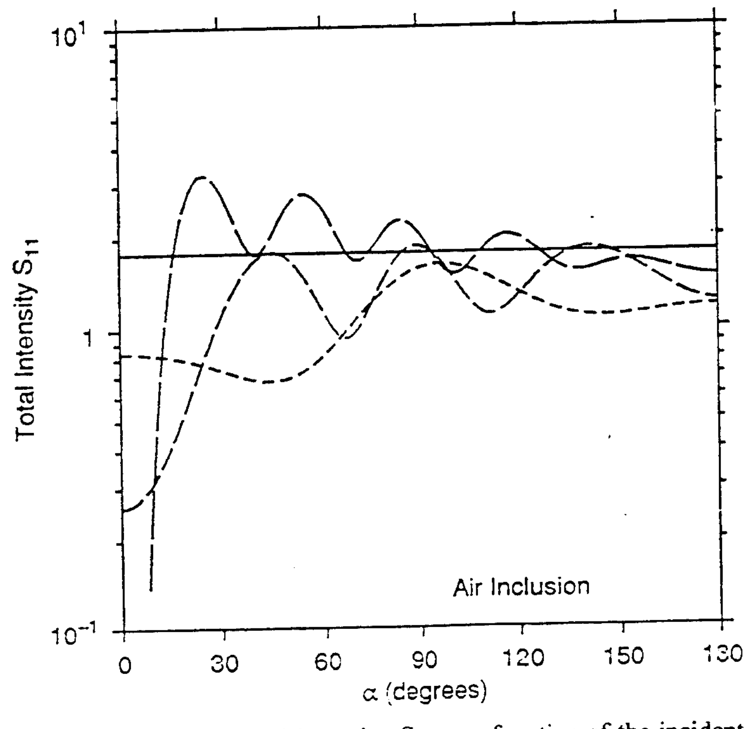


Figure 7. The backscattering total intensity S_{11} as a function of the incident angle α for a perfectly conducting inclusion (top) and an air inclusion (bottom) inside an $a_1 = \lambda$ water host ($\tilde{m}_1 = 1.33 + 0i$). The inclusion radius is $a_2 = \lambda/4$.

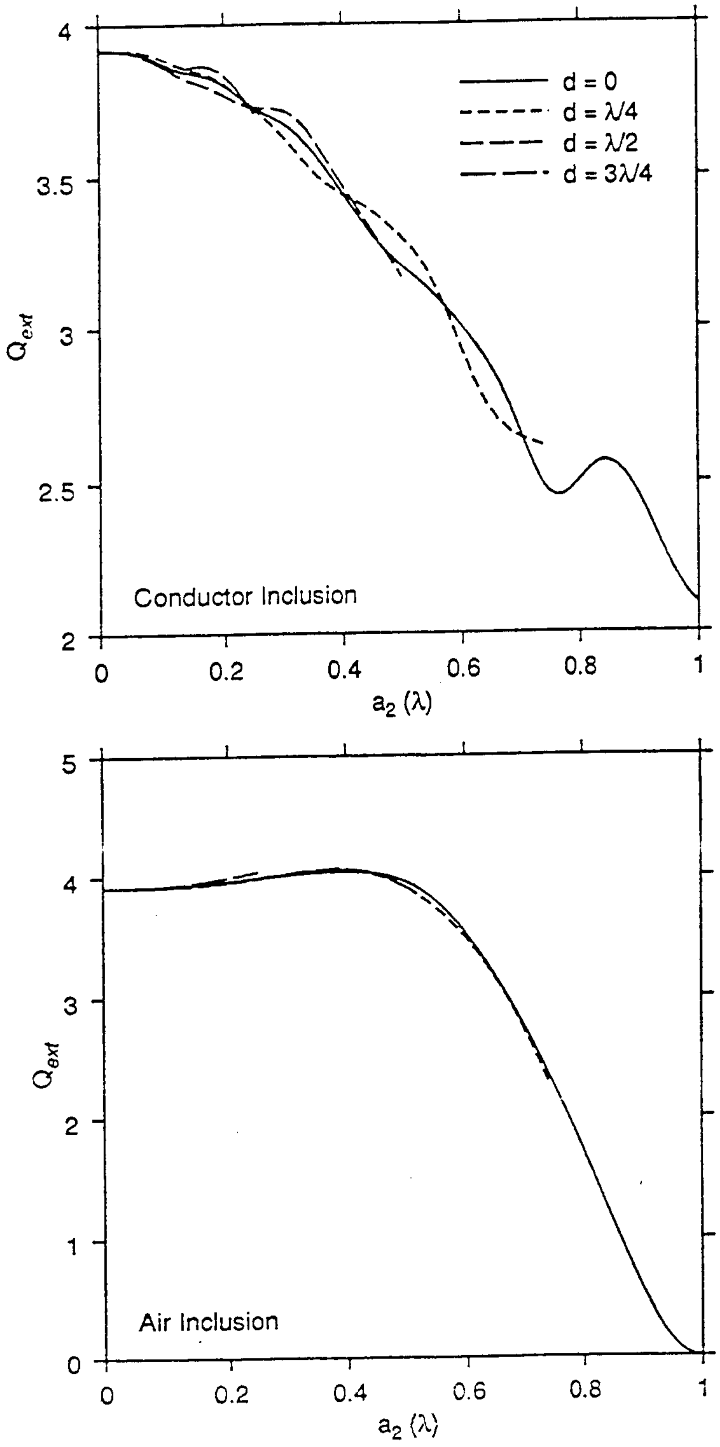
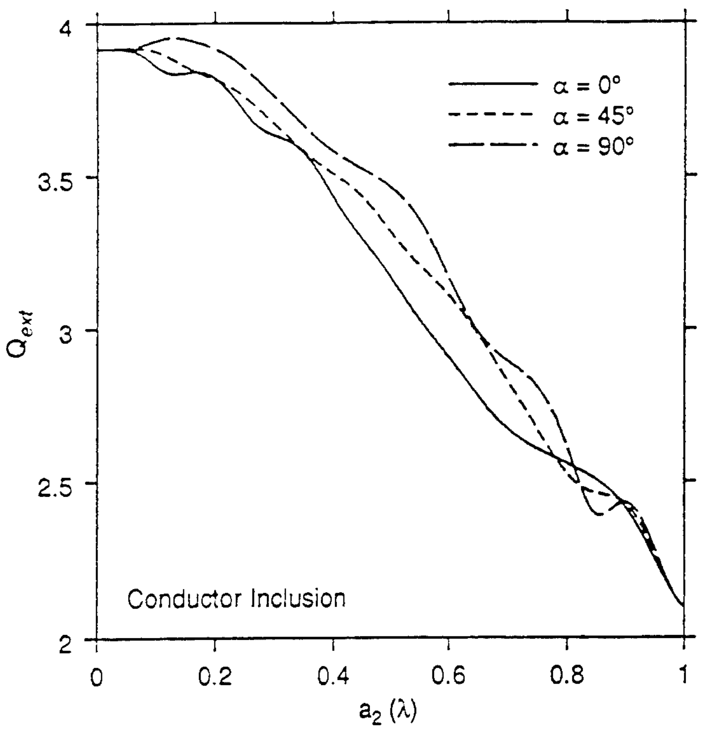


Figure 8. The extinction efficiencies as a function of inclusion radius when the incident angle $\alpha = 0°$ for a perfectly conducting inclusion (top) and an air inclusion (bottom) inside an $a_1 = \lambda$ water host ($\tilde{m}_1 = 1.33 + 0i$).



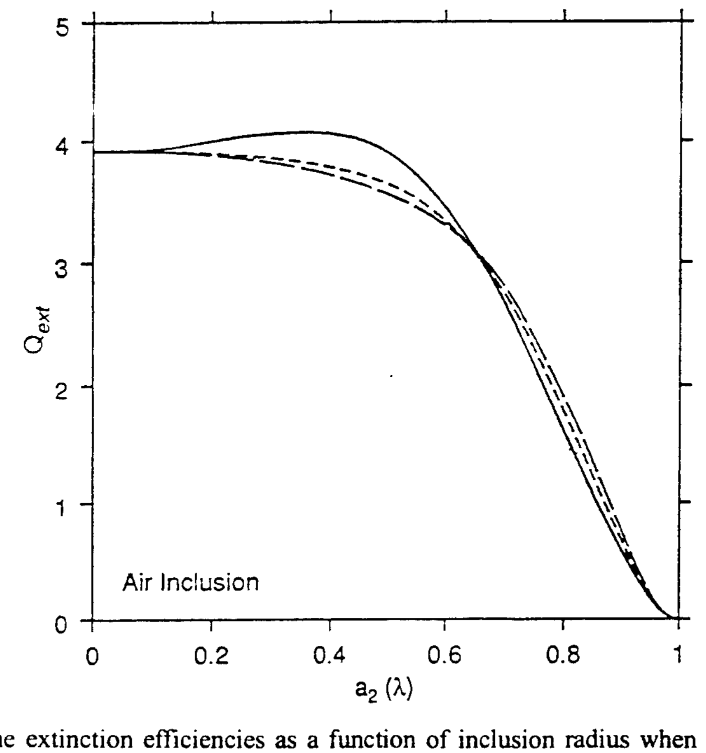


Figure 9. The extinction efficiencies as a function of inclusion radius when the incident angle $\alpha = 0°$ and $a_1 = a_2 + d$ for a perfectly conducting inclusion (top) and an air inclusion (bottom) inside an $a_1 = \lambda$ water host ($\tilde{m}_1 = 1.33 + 0i$).

different separation distances when $\alpha = 0^0$. The solid lines in both graphs (d = 0) correspond to the cases where the host and inclusion spheres are concentric. We also calculated the extinction efficiencies for systems having negative values of the displacements shown, but, as has been discussed earlier, these values are identical to the extinction efficiencies of systems having positive displacements.

Figure 9 shows the extinction efficiencies as a function of the inclusion radius when the inclusion is at the edge of the host sphere ($a_1 = a_2 + d$) when $\alpha = 0^0$. From system symmetry and the reciprocity theorem, $Q_{ext}(\alpha) = Q_{ext}(-\alpha) = Q_{ext}(180^0 - \alpha)$. A comparison of Figures 8 and 9 shows that there is a greater range in extinction efficiencies, and therefore a greater sensitivity of the efficiencies, when the angle of incidence is varied than when the inclusion is translated in the direction of the incident beam.

Examinations of the above figures reveal two important findings for the nonconcentric sphere system. The first finding is that the extinction efficiency is independent of the sign of the displacement of the inclusion particle. In other words, the system gives the same value of the extinction efficiency regardless of whether the inclusion lies in the forward direction (where the fields are generally stronger) or in the backscatter direction (where the fields are generally weaker). We also see that the closer an inclusion is to the surface of the host sphere, the greater the effect it has on the extinction of the system. This last point is very important in terms of resonance suppression.

5 CONCLUSION

We have derived solutions for the total field when an incident plane wave strikes a host sphere containing a nonconcentric spherical inclusion. Reduction of the general result to that of the concentric sphere and the Mie sphere verifies the validity of the derivation. We have examined the scattering as a function of a number of system parameters. Of particular interest is the independence of the extinction efficiency on the sign of the displacement of the inclusion particle. We also see that the closer an inclusion is to the surface of the host sphere, the greater the effect it has on the extinction of the system.

Blank

LITERATURE CITED

[1] J. Goldenson and J.D. Wilcox, "Carrier dusts for toxic aerosols. I. Preliminary survey of dusts," **TCR-66**, Chemical Corps Technical Command, Army Chemical Center (1950).

[2] G.H. Milly and R.M. Black, "Report of field test 266, static test of a single 10lb. experimental bomb filled GB on carrier dust," **TCIR-581**, Chemical Corps Technical Command, Army Chemical Center (1950).

[3] A.L. Aden and M. Kerker, "Scattering of electromagnetic waves from two concentric spheres," J. Appl. Phys. **22**, 1242-1246 (1951).

[4] R.W. Fenn and H. Oser, "Scattering properties of concentric soot-water spheres for visible and infrared light," Appl. Opt. **4**, 1504-1509 (1965).

[5] K. A. Fuller, "Scattering of light by coated spheres," Opt. Lett. **18**, 257-259 (1993).

[6] D.S. Wang and P.W. Barber, "Scattering by inhomogeneous nonspherical objects," Appl. Opt. **18**, 1190-1197 (1979).

[7] D. S. Wang, **Light Scattering by Nonspherical Multilayered Particles**, Ph.D. dissertation, Dept. of Physics, U. of Utah, Salt Lake City (1979).

[8] D.Q. Chowdhury, S.C. Hill, and P.W. Barber, "Morphology-dependent resonances in radially inhomogeneous spheres," J. Opt. Soc. Am. A **8**, 1702-1705 (1991).

[9] P. Chýlek, V. Srivastava, R.G. Pinnick, and R.T. Wang, "Scattering of electromagnetic waves by composite spherical particles: experiment and effective medium approximations," Appl. Opt. **27**, 2396-2404 (1988).

[10] B. Friedman and J. Russek, "Addition theorems for spherical waves," Quart. Appl. Math. **12**, 13-23 (1954).

[11] S. Stein, "Addition theorems for spherical wave functions," Quart. Appl. Math. **19**, 15-24 (1961).

[12] O.R. Cruzan, "Translational addition theorems for spherical vector wave functions," Quart. Appl. Math. **20**, 33-40 (1962).

[13] J.G. Fikioris and N.K. Uzunoglu, "Scattering from an eccentrically stratified dielectric sphere," J. Opt. Soc. Am. **69**, 1359-1366 (1979).

[14] F. Borghese, P. Denti, R. Saija, and O.I. Sindoni, "Optical properties of spheres containing a spherical eccentric inclusion," J. Opt. Soc. Am. A **9**, 1327-1335 (1992).

[15] K.A. Fuller, "Scattering and absorption by inhomogeneous spheres and sphere aggregates," SPIE Proc. **1862**. 249-257 (1993).

[16] P.A. Bobbert and J. Vlieger, "Light scattering by a sphere on a substrate," Physica **137A**,

209-241 (1986).

[17] G. Videen, **Light Scattering from a Sphere on or Near an Interface,** Ph.D. dissertation, Dept. of Physics, U. of Arizona, Tucson (1992).

[18] D. Ngo, **Light Scattering from a Sphere with a Nonconcentric Spherical Inclusion**, Ph.D. dissertation, Dept. of Physics, New Mexico State University, Las Cruces (1994).

[19] C.F. Bohren and D.R. Huffman, **Absorption and Scattering of Light by Small Particles** (Wiley, New York, 1983).

Appendix: **NONCONCENTRIC SPHERE CODE**

Description of The Program

This appendix contains the FORTRAN code for the nonconcentric spheres. The program consists of the main routine and twelve subroutines. Most calls to the subroutines are made from the main program. We will briefly describe each subroutine and its respective variables. This program calculates the Mueller scattering matrix elements (S_{ij}), and the extinction (Q_{ext}), scattering (Q_{sca}), and absorption (Q_{abs}) efficiencies for a set of initial inputs. The input data is read from the "input.in" file. All the variables used are defined in a separate file named "declare.def". The reason for using this method is the convenience of having all the variables in one separate file for comparison and debugging purposes. Explanation of some important variables are given in the subsections below. Also, a short descriptions of the subroutines are presented.

The only restriction placed on this program is that we cannot allow the inclusion radius to equal zero, i.e. $a_2 \neq 0$. The reason is that the Neumann functions blow up if the argument is zero. If the user wishes to have no inclusion, the user can let the index of refraction of the inclusion equal that of the host sphere, i.e. $\tilde{m}_1 = \tilde{m}_2$.

Subroutine bessel (n,np,bessj,x)

This subroutine calculates the spherical Bessel function of the first kind from order -1 to order n, and stores the results in the array bessj. The variable n is related to the size parameter x. If x is large, then n is also large. The variable np is the maximum size (the physical dimension) of the array. Here np is isize, where isize is defined in the file "declare.def"

Input

n: The largest order of the Bessel function, which is determined in the main routine by the definition of nbg.

np: The array size defined by the parameter isize.

x: The argument of the Bessel function.

Output

bessj: This double precision real array contains the values of the spherical Bessel function of the first kind $[j_n(k_o a_1)]$, from order -1 to order n.

Subroutine c_bessel(n,np,bessj,x)

This subroutine is similar to the Bessel subroutine except that it takes a complex argument x instead of a real argument. The array bessj is a complex double.

Input

n: The largest order of the Bessel function, which is determined in the main routine by the definition of nbg.

np: The array size defined by the parameter isize.

x: The complex argument of the Bessel function.

Output

bessj: This double complex array contains the complex values for this complex spherical Bessel function of the first kind $[j_n(k_1 a_1)]$. from order -1 to order n.

Subroutine newman(n,np,bessy,x)

This subroutine calculates the spherical Bessel function of the second kind (or the Neumann function, $n_\nu(k_o a_1)$), from order -1 to order n and stores the results in the double precision array bessy.

Input

n: The largest order of the Neumann function, which is determined in the main routine by the definition of nbg.

np: The array size defined by the parameter isize.

x: The argument of the Neumann function.

Output

bessy: This double precision real array contains the values for this spherical Neumann function $[n_\nu(k_o a_1)]$, from order -1 to order n.

Subroutine c_newman(n,np,bessy,x)

This subroutine is similar to the newman subroutine, except that it takes a complex argument x instead of a real argument. The array bessy is now a complex double.

Input

n: The largest order of the Neuman function, which is determined in the main routine by the definition of nbg.

np: The array size defined by the parameter isize.

x: The complex argument of the Neumann function.

Output

bessj: This double complex array contains the complex values for this spherical Neumann function $[n_\nu(k_1 a_1)]$, from order -1 to order n.

Subroutine riccati(nbg,hval,zeta,dzeta,x)

This subroutine takes as input the Hankel functions of the first or second kind and transform them into the Riccati-Bessel functions and their derivatives. The transformation follows from the definition of the Riccati-Bessel functions shown in the preceding derivations.

Input

nbg: The largest order for the Hankel and Riccati-Bessel functions.

hval: The double complex array which contains either the first $[h_n^1(k_1 a_1)]$ or the second $[h_n^2(k_1 a_1)]$ Hankel functions as defined and computed in the main routine. The values stored in this array range from order -1 to order nbg.

x: The double complex argument of the spherical Hankel function.

Output

zeta: This is the Riccati-Bessel function. The values are stored from order zero to order nbg into this double complex array.

dzeta: This is the derivative of the Riccati-Bessel function, the results are stored into this double complex array.

Subroutine nplgndr(m,nbr,iisize,legpol,x)

Given a value for m, where m $\leq$ nbr, this subroutine calculates the normalized associated Legendre polynomials $\left[\tilde{P}_n^m(x)\right]$ from order $m = 0$ to nbr and stores the values in the array legpol.

Input

m: The integer order for legpol, which ranges from 0 to nbg.

nbr: This variable has a maximum value of nbg+1; the iteration of this subroutine goes from m to nbr.

iisize: This variable is defined as isize + 1 (shown in the "declare.def" file).

x: This argument for the normalized associated Legendre polynomials is the cosine of the angle, i.e. $x = cos\theta$.

Output

legpol: This is the one-dimensional array containing the calculated values. It has a physical dimension of isize + 1.

Subroutine ludcmp(a,n,np,indx)

This subroutine decomposes and Tri-diagonalizes a matrix. Given a matrix a(1:n,1:n) with physical dimension np, where np is defined as isiz2 in the "declare.def" file, this subroutine replaces it by the LU Decomposition of a row-wise permutation of itself. The procedure is an n^3 process, so that for large size parameters, the computation time slows down dramatically. If nbg is bigger than 70 (which corresponds to a size parameter of $x \approx 50$), one should change the value of nmax in this subroutine so that one always has $nmax > 2(nbg + 1)$. This routine is used in conjunction with the subroutine lubksb[1] to solve linear equations or invert a matrix. Note that we have modified this subroutine to accept double complex arguments.

Input

a: This is a double complex $n \times n$ matrix given by equations 40 and 41 in the preceding derivation. After this matrix passes through this subroutine, it will be destroyed and replaced with its tri-diagonal.

n: This integer value is equal to 2(nbg+1). The parameter nmax, defined within this subroutine, must be greater than or equal to this value.

np: This is the physical dimension size of the matrix and has a value of isiz2. It is safe to let it be larger than n.

Output

a: The original matrix is now a tri-diagonalized matrix. This new matrix can now be used in conjunction with the subroutine lubksb to solve the linear equation.

indx: This is an array which records the row permutation effected by the partial pivoting.

Subroutine lubksb(a,n,np,indx,b)

This subroutine solves the set of n linear equations $ax = b$. The solution vector is represented by the array b, and the solution vector x, after going through the subroutine, is also stored in b. The variables n, np, and the arrays a and indx, are not modified by this routine and can be left in place for successive calls with different right-hand sides b. This routine takes into account the possibility that b will begin with many zero elements, so it is efficient for use in matrix inversion. We have also modified this subroutine to accept double complex arguments.

Input

a: This double complex matrix, with physical dimension np, is the same matrix a which has undergone the tri-diagonalization in the subroutine ludcmp.

n: This integer value is equal to 2(nbg+1). The variable nmax, defined within this subroutine, must be greater than or equal to this value.

np: This is the physical dimension size of the matrix and has a value of isiz2. We must keep in mind that np must be greater than n.

indx: This is the output vector which records the row permutation effected by the partial

[1]Numerical Recipes, 2nd Edition

pivoting in the subroutine ludcmp.

b: This input vector is the right-hand side of the equation $ax = b$, i.e. it is the solution vector. The subroutine will solve for the vector x, and places its values in b.

Output

b: The vector x is now placed in this array.

Subroutine mprove(a,alud,n,np,indx,b,x)

This subroutine, which has been modified to accept double complex arguments, improves a solution vector x of the linear set of equations $ax = b$. It assumes that the vector x is off from the true solution by δx, and it recursively solves for δx and improves that vector. It does this by comparing the two matrices a and alud, with the solution b and argument x, and improves on the solution of x. Note that inside this subroutine is the parameter nmax, which should have the same value as that given in the subroutine ludcmp.

Input

a: This $n \times n$ matrix is the original matrix before it has undergone any changes due to the result of the subroutine ludcmp. All the values are in their original state.

alud: This $n \times n$ matrix is the one that underwent the tri-diagonalization associated with the ludcmp subroutine. It is the output matrix, a, given in the ludcmp subroutine above.

n: The size of the two matrices, and the size of the vector x, b, and indx.

np: The physical dimension of the two matrices.

indx: The output vector returned by the subroutine ludcmp.

b: The solution vector of the equation $ax = b$.

x: This vector is the output from the subroutine lubksb.

Output

x: After mprove, x has been modified and improved to a new set of values. Note that this subroutine can be called as often as one wishes, but generally, one or two calls are enough.

Function find_g(nbg,nn,ang,cte,dte,ctm,dtm)

This function calculates the asymmetry parameter, g, and returns the value in double precision to the main routine.

Input

nbg: The largest order of the Bessel function defined in the main routine.

nn: The physical size of the arrays used.

ang: The incident angle.

cte: This is the array which contains the scattering coefficients for the TE mode.

dte: This too contains the scattering coefficients for the TE mode.

ctm: This array contains the scattering coefficients for the TM mode.
dtm: This contains the scattering coefficients for the TM mode.
Output
fing_g: Returns as a double precision real number.

Subroutine rotation(a,b,c,d,nbg,ang,cte,dte,ctm,dtm,nn,ij)

This subroutine is used in conjunction with the function find_g to calculate the asymmetry parameter. We must rotate and integrate over the entire sphere in order to find the asymmetry parameter.
Input
nbg: The largest order of the Bessel function used.
ang: The incident angle.
nn: The physical size of the arrays sent from the main routine.
ij: The physical size of the arrays sent from the function find_g.
cte, dte, ctm, dtm: The arrays which contain the scattering coefficients as sent in from the main routine. These values are now stored into the arrays a, b, c, d respectively.
Output
a, b, c, d: These are the new rotated arrays containing the scatttering coefficients. Now we can use these new arrays to calculate the asymmetry parameter.

Function factorial(n)

This function calculates the factorial of a given integer. Since a factorial of a large integer will blow up, we define this function to take the sum of the log of the number. This will ensure us of stability in our calculations.
Input
n: The integer value sent in from the subroutine rotation.
Output
factorial: This function now returns a double precision real number.

Declare.def

Within the main routine is an include statement which refers to this file. The parameter isize is presently set to handle a size parameter of up to 50. The program can be made to handle an even larger size parameter by increasing isize appropriately. Note that if the size parameter of the run is greater than 50, one must change isize to a larger number to include all memory allocations; also, the parameter nmax in the subroutines ludcmp and mprove must be increased in value. The other parameter, isiza, is presently set to handle up to 360 degrees by increments of 9 degrees. If the run is to have a resolution greater than 1 degree (which can be changed in the input file with the variable num), isiza must be changed to meet that specific need.

```
c          ..................................................................................
c          . Scattering program for an inclusion imbedded inside the   .
c          . host droplet.  The input data is taken from the file         .
c          . "input.in" and the variables and arrays are defined in      .
c          . the file "declare.def".                                       .
c          ..................................................................................

           implicit double precision (a-h,o-z)
           INCLUDE 'declare.def'

           open(unit=4,file='input.in',status='old')
           rewind(4)
           open(unit=8,file='output.out')
c .............................................................
c . read in the input values from "input.in"  .
c .............................................................
           read(4,*) wavel
           read(4,*) rad1
           read(4,*) rad2
           read(4,*) distance
           read(4,*) m_r1,m_i1
           read(4,*) m_r2,m_i2
           read(4,*) inc_ang
           read(4,*) num
           read(4,*) phi

           pi=1.0
           ccc=dcmplx(0.0,pi)
           pi=4.0*datan(pi)

       m_1=dcmplx(m_r1,m_i1)
       m_2=dcmplx(m_r2,m_i2)
       k0=2.*pi/wavel
       k1=k0*m_1
       k2=k0*m_2
       x0=k0*rad1
       x1=k1*rad1
       x2=k1*rad2
       x3=k2*rad2

c      nbg= # of iterations for the big sphere and loops
       nbg=cdabs(x0) + 4.*cdabs(x0)**.333 + 2.0
       print*,'the value of nbg=',nbg
```

```
c ..........................................................................................
c .     Calculate Bessel Functions for the first boundary at rad1                .
c .      We have bessel as an incident field of the sphere, and a                .
c .      hankel as the scattered field.  Their arguments are x0 and cx0          .
c .      Then we have hankel1 and hankel2 inside the sphere; and                 .
c .      their argument is k1*rad1=x1.                                           .
c ..........................................................................................
      cx0=dreal(x0)
      call bessel(nbg,isize,besj_o,cx0)
      call newman(nbg,isize,temp,cx0)
      call c_bessel(nbg,isize,hankel_o,x0)
      do 11 i=-1,nbg
       hankel_o(i)=hankel_o(i)+ccc*temp(i)
 11   continue

      call c_bessel(nbg,isize,hankel1_1,x1)
      call c_bessel(nbg,isize,hankel2_1,x1)
      call c_newman(nbg,isize,tempc,x1)
      do 12 i=-1,nbg
         hankel1_1(i)=hankel1_1(i) + ccc*tempc(i)
         hankel2_1(i)=hankel2_1(i) - ccc*tempc(i)
 12   continue

c ..........................................................................................
c .      Now we match boundary at rad2.  Here we have two hankels in              .
c .      the host sphere, and their argument is k1*rad2=x2                        .
c .      Then we have a bessel inside the inclusion with its argument             .
c .      being x3.  Note: since this is for the small inclusion, we               .
c .      must take care to have spherical bessel functions which are              .
c .      within the limits of that size; the order should never exceed            .
c .      the size of the sphere because it'll blow up in your face                .
c ..........................................................................................
      nsm=cdabs(x2) + 4.*cdabs(x2)**.333 + 2.0
      if(m_r2.gt.0.0) call c_bessel(nsm,isize,besj_2,x3)
      call c_bessel(nsm,isize,hankel1_2,x2)
      call c_bessel(nsm,isize,hankel2_2,x2)
      call c_newman(nsm,isize,tempc,x2)
      do 13 i=-1,nsm
         hankel1_2(i)=hankel1_2(i) + ccc*tempc(i)
         hankel2_2(i)=hankel2_2(i) - ccc*tempc(i)
 13   continue

c ..........................................................................................
c .   Calculate Riccati-Bessel Functions and Derivatives needed                   .
c .      Note that zeta1_1 means the 1st zeta at k1a1                             .
```

```
c .     and      zeta2_1 means the 2nd zeta at k1a1           .
c .     and      zeta1_2 means the 1st zeta at k1a2           .
c ...........................................................
        psi_o(-1)=cx0*besj_o(-1)
        do 21 n=0,nbg
             psi_o(n)=cx0*besj_o(n)
             dpsi_o(n)=cx0*besj_o(n-1) -n*besj_o(n)
 21     continue
        call riccati(nbg,hankel_o,zeta_o,dzeta_o,x0)
        call riccati(nbg,hankel1_1,zeta1_1,dzeta1_1,x1)
        call riccati(nbg,hankel2_1,zeta2_1,dzeta2_1,x1)
        call riccati(nsm,hankel1_2,zeta1_2,dzeta1_2,x2)
        call riccati(nsm,hankel2_2,zeta2_2,dzeta2_2,x2)
        call riccati(nsm,besj_2,psi_2,dpsi_2,x3)

c ..........................................................................
c .     Now we must solve for the Quality factors by using equations      .
c .     17 and 18; These Q's relate the inclusion's field coeffs          .
c .     to the outter sphere's coefficients, and it contains infos such   .
c .     as the inclusions's radius and refractive index                   .
c .    Note that if the index of refraction of the inclusion is nega-     .
c .     tive. then we'll take that to mean it is a perfect conductor.     .
c .     and the simplification is in the "else" part of the if-else       .
c .     statement.  We cannot use index > 100 because it'll blow up       .
c ..........................................................................
        if(m_r2.gt.0.0) then
         do 25 n=0,nsm
          fact1=k1*dzeta2_2(n)*psi_2(n)-k2*zeta2_2(n)*dpsi_2(n)
          fact2=k2*zeta1_2(n)*dpsi_2(n)-k1*dzeta1_2(n)*psi_2(n)
          Q_r(n)=fact1/fact2
          fact3=k2*dzeta2_2(n)*psi_2(n)-k1*zeta2_2(n)*dpsi_2(n)
          fact4=k1*zeta1_2(n)*dpsi_2(n)-k2*dzeta1_2(n)*psi_2(n)
          Q_s(n)=fact3/fact4
 25      continue
c  Now to fill in the rest of the array with a constant value.  This is a must.
         do 26 n=nsm+1,nbg
          Q_r(n)=dcmplx(1.0,0.0)
          Q_s(n)=Q_r(n)
 26      continue
        else
c ..........................................
c . The inclusion is a perfect conductor  .
c ..........................................
          print*,'The inclusion is a perfect conductor'
          do 35 n=0,nsm
```

```
            Q_r(n)=-zeta2_2(n)/zeta1_2(n)
            Q_s(n)=-dzeta2_2(n)/dzeta1_2(n)
35         continue
           do 36 n=nsm+1,nbg
            Q_r(n)=dcmplx(1.0,0.0)
            Q_s(n)=Q_r(n)
36         continue
          endif
c ..........................................................................................
c .   Calculate translation coefficients when the separation of the    .
c .   center to center radii is d.  We will first find the bessel       .
c .   function which describes the separation, k1d                      .
c ..........................................................................................
          xd=(k0*m_1)*distance
          if (xd.eq.0.0) then
                  nsm=2
          else
                  nsm=cdabs(xd) + 4.*cdabs(xd)**.333 +2.
          endif
          call c_bessel(3*nsm,isiz4,besj_d,xd)

c ..................................................................
c .   Begin the scalar translation coefficients .
c .   keeping in mind that cuplo = c0,0(np)     .
c .   and cuplo1 = c-1,0(np) for starters       .
c ..................................................................

      do 110 np=0,nbg*4
         fnp=dfloat(np)
       cuplo(np)=besj_d(np)*dsqrt(2.*fnp+1.)
       cuplo1(np)=-cuplo(np)
 110  continue
      do 115 np=0,nbg*2
       cuplom(0,np)=cuplo(np)
 115  continue

c .......................................................................
c .    Now for some acrobatics; this is where we .
c .       perform the switcharoos and recursively .
c .       solve for the translation coeff.  The   .
c .       result is stored into the matrix cuplom .
c .......................................................................
       do 140 n=1,nbg*2
          fn=dfloat(n)
        do 120 np=0,nbg*4-n
```

```
          fnp=dfloat(np)
        c=cdsqrt(dcmplx((2.*fnp+1.)/(2.*fn-3.)))*(fn-1.)
        c=cuplo1(np)*c
        cuplo1(np)=cuplo(np)
        cuplo(np)=c
 120    continue

       do 130 np=0,nbg*4-n
          fnp=dfloat(np)
        c=-(fnp+1.)*dsqrt((2.*fn-1.)/(2.*fnp+3.))*cuplo1(np+1)
      c=c+fnp*cdsqrt(dcmplx((2.*fn-1.)/(2.*fnp-1.)))*cuplo1(np-1)
        c=c+cuplo(np)
        cuplo(np)=c*dsqrt((2.*fn+1.)/(2.*fnp+1.))/fn
        if (np.le.nbg*2) then
         cuplom(n,np)=cuplo(np)
c        write(8,*)n,np,cuplom(n,np)
        endif
 130    continue
 140  continue

c ........................................................................
c . Now that we have the elements in the matrix          .
c . cuplom(n,np), we can start the iterations in          .
c . m to get the rest of the translation coeff            .
c . Then we'll have the complete set of C(n,m)np          .
c ........................................................................
        do 299 m=0,nbg
c       ......................................................................
c       . But first we need to calculate, for each m, the   .
c       . Legendre Polynomials for incident angle theta  .
c       ......................................................................
        ii=nbg+1
        x=dcos(pi*inc_ang/180.)
        mmm=m+1
        call nplgndr(mmm,ii,isize1,legpol1,x)
        mmm=m
        call nplgndr(mmm,ii,isize1,legpol2,x)
        if (m.gt.0) then
          mmm=m-1
          call nplgndr(mmm,ii,isize1,legpol,x)
        else
          do 165, jjj=-2,isize1
            legpol(jjj)=-legpol1(jjj)
165         continue
        endif
```

```
          do 180 i=m,nbg
               fi=dfloat(i)
            tau=dsqrt((fi+m+1.)*(fi-m))*legpol1(i)
            pie=tau
            tau=(tau-dsqrt((fi+m)*(fi-m+1.))*legpol(i))/2.0
            if (dabs(x).gt.0.5) then
c         ......................................................................
c         . The angle theta is between 0 and 60 degrees          .
c         ......................................................................
              pie=pie+dsqrt((fi+m)*(fi-m+1.))*legpol(i)
              pie=pie*.5/x
            else
c         ......................................................................
c         . theta is between 60 and 90; use this so the          .
c         . Polynomial doesn't blow up at 90 or 0 degrees        .
c         ......................................................................
              pie=m*legpol2(i)/dsin(inc_ang*pi/180.)
            endif
            if (i.gt.0) then
                  c = (ccc**i)/(fi*(fi+1.))
                  ba(i)=-c*pie
                  aa(i)=c*tau
            endif
 180        continue
c ..............................................................................
c .       That ends the Calculation for the Legendre Polynomials      .
c .       for each value of m.  The incident field coefficients       .
c .       for the TE case are stored in aa(i) and ba(i); the TM       .
c .       case will be done later by flipping the TE coefficients     .
c ..............................................................................


c ..............................................................................
c . Now to find the rest of the matrix in C(n,m)np.  For each         .
c . value of m that is greater than 0, say 1, we can find the         .
c . matrix C(n,1)np and store that into cuplom(n,np); then            .
c . the next iteration of m, i.e. m=2, we can find C(n,2)np           .
c . by already having C(n,1)np stored in the matrix cuplom(n,np)      .
c ..............................................................................
      if (m.gt.0) then
          do 155, n=m-1,2*nbg-m+1
          do 154, np=m-1,2*nbg-m+1
                        cuplom1(n,np)=cuplom(n,np)
 154              continue
 155      continue
```

```
        do 160 n=m,2*nbg-m
          fn=dfloat(n)
       do 159 np=m,2*nbg-m
           fnp=dfloat(np)
     c=dsqrt((fnp-m+1.)*(fnp+m)*(2.*fnp+1.))*cuploml(n,np)
        cuplom(n,np)=c
        c=xd*dsqrt((fnp-m+2.)*(fnp-m+1.)/(2.*fnp+3.))
        cuplom(n,np)=cuplom(n,np)-c*cuploml(n,np+1)
        c=xd*dsqrt((fnp+m)*(fnp+m-1.)/(2.*fnp-1.))
        cuplom(n,np)=cuplom(n,np)-c*cuploml(n,np-1)
        c=dsqrt((fn-m+1.)*(fn+m)*(2.*fnp+1.))
        cuplom(n,np)=cuplom(n,np)/c
 159      continue
 160     continue
      endif
c ..................................................................
c . Cuplom(n,np) now has the most recent values   .
c . for the most recent value of m.  Note that      .
c . we didn't find C(n,0)np because we already      .
c . have those values stored in the cuplom matrix   .
c ..................................................................

c .................................................................................
c . Now that we have the matrix which contains the C(n,m)np  .
c . we can go ahead and find the vector translation coefficients .
c . A(n,m)np and B(n,m)np                                      .
c .................................................................................
      do 190 n=m,nbg
         pie=0.0
         tau=0.0
         do 195 np=m,nbg
           fnp=dfloat(np)
           tau=(fnp-m+1.0)*(fnp+m+1.0)
           tau=tau/((2.*fnp+1.)*(2.*fnp+3.))
           tau=dsqrt(tau)
           TransA=-xd/(fnp+1.0)
        TransA=TransA*tau*cuplom(n,np+1)
        TransB=0.0
        if (np.gt.0) then
           pie=(fnp-m)*(fnp+m)
           pie=pie/((2.*fnp-1.)*(2.*fnp+1.))
           pie=dsqrt(pie)
           c=-xd*pie*cuplom(n,np-1)/fnp
         TransA=TransA+c
         TransB=-ccc*xd*m*cuplom(n,np)/(fnp*(fnp+1.))
        endif
        TransA=cuplom(n,np)+TransA
          TransAm(n,np)=TransA
          TransBm(n,np)=TransB
195     continue
190  continue

     numel=nbg-m+1
     numtot=numel+numel
     do 220 n=m,nbg
        i=n-m+1
        do 215 np=m,nbg
c ..............................................................................
c . The Q's have the index of the inclusion k2 inside them              .
c . and the TransAm and TransBm have the displacement d inside           .
c . them; matrixlu therefore is the 1st to contain both pieces           .
c . of information.                                                      .
c ..............................................................................
            fact1=dzeta_o(n)*(zeta2_1(n)+Q_r(np)*zeta1_1(n))
            fact2=zeta_o(n)*(dzeta2_1(n)+Q_r(np)*dzeta1_1(n))
            fact3=dzeta_o(n)*(zeta2_1(n)+Q_s(np)*zeta1_1(n))
            fact4=zeta_o(n)*(dzeta2_1(n)+Q_s(np)*dzeta1_1(n))
          ii=np-m+1

          c=TransAm(np,n)*(k0*fact1 - k1*fact2)
          matrixlu(i,ii)=c
          matrix(i,ii)=c

          c=TransBm(np,n)*(k0*fact3 - k1*fact4)
          matrixlu(i,ii+numel)=c
          matrix(i,ii+numel)=c

          c=TransBm(np,n)*(k1*fact1 - k0*fact2)
          matrixlu(i+numel,ii)=c
          matrix(i+numel,ii)=c

          c=TransAm(np,n)*(k1*fact3 - k0*fact4)
          matrixlu(i+numel,ii+numel)=c
          matrix(i+numel,ii+numel)=c
215       continue
220  continue

c..............................................................................
c. THE MATRIX IS FILLED!!  That was the heart of the problem .
c..............................................................................
```

```
c ..........................................................................................
c . Now we can load up the solution vectors aa(n) and ba(n)          .
c . and start the LU Decomposition to get t(n) and u(n)              .
c . Note that aa(n) and ba(n) contain the incident angle             .
c ..........................................................................................
c ...................
c For TE case .
c ...................
        do 226 n=m,nbg
         i=n-m+1
         c=dzeta_o(n)*psi_o(n)
            c = k1*(c - dpsi_o(n)*zeta_o(n))
         b(i)=aa(n)*c
         b(i+numel)=ba(n)*c
         bd(i)=b(i)
         bd(i+numel)=b(i+numel)
 226      continue
        ii=isiz2
        call ludcmp(matrixlu,numtot,ii,indx,dd)
        call lubksb(matrixlu,numtot,ii,indx,b)
        call mprove(matrix,matrixlu,numtot,ii,indx,bd,b)
           do 227 n=m,nbg
                i=n-m+1
                TintTE(m,n)=b(i)
                UintTE(m,n)=b(i+numel)
 227       continue

c ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
c .  Now we must find the Scattering coefficients CscaTE and DscaTE,        .
c .  and from these coeff we can determine the scattering matrices.         .
c ..............................................................................................
        do 250 n=m,nbg
          sum1=0.0
          sum2=0.0
          do 260 np=m,nbg
            fact1=dzeta2_1(n)+Q_r(np)*dzeta1_1(n)
            fact1=TransAm(np,n)*TintTE(m,np)*fact1
            fact2=dzeta2_1(n)+Q_s(np)*dzeta1_1(n)
            fact2=TransBm(np,n)*UintTE(m,np)*fact2
            sum1=fact1+fact2+sum1
            fact3=zeta2_1(n)+Q_r(np)*zeta1_1(n)
            fact3=TransBm(np,n)*TintTE(m,np)*fact3
            fact4=zeta2_1(n)+Q_s(np)*zeta1_1(n)
            fact4=TransAm(np,n)*UintTE(m,np)*fact4
            sum2=sum2+fact3+fact4
 260      continue
          CscaTE(m,n)=(sum1-aa(n)*dpsi_o(n))/dzeta_o(n)
          DscaTE(m,n)=(sum2-ba(n)*psi_o(n))/zeta_o(n)
c       print*,n,CscaTE(m,n),DscaTE(m,n)
 250    continue

c..............................
c FOR TM CASE!!! .
c..............................
        do 228 n=m,nbg
          i=n-m+1
          c=dzeta_o(n)*psi_o(n)
            c =k1*(c - dpsi_o(n)*zeta_o(n))
          b(i)=ccc*ba(n)*c
          b(i+numel)=ccc*aa(n)*c
          bd(i)=b(i)
          bd(i+numel)=b(i+numel)
 228      continue
        ii=isiz2
        call lubksb(matrixlu,numtot,ii,indx,b)
        call mprove(matrix,matrixlu,numtot,ii,indx,bd,b)
        do 229 n=m,nbg
           i=n-m+1
          TintTM(m,n)=b(i)
          UintTM(m,n)=b(i+numel)
 229     continue

c +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
c .  Now we must find the Scattering coefficients CscaTM and DscaTM,     .
c .  and from these coeff we can determine the scattering matrices.      .
c ..............................................................................
        do 251 n=m,nbg
          sum1=0.0
          sum2=0.0
          do 261 np=m,nbg
            fact1=dzeta2_1(n)+Q_r(np)*dzeta1_1(n)
            fact1=TransAm(np,n)*TintTM(m,np)*fact1
            fact2=dzeta2_1(n)+Q_s(np)*dzeta1_1(n)
            fact2=TransBm(np,n)*UintTM(m,np)*fact2
            sum1=fact1+fact2+sum1
            fact3=zeta2_1(n)+Q_r(np)*zeta1_1(n)
            fact3=TransBm(np,n)*TintTM(m,np)*fact3
            fact4=zeta2_1(n)+Q_s(np)*zeta1_1(n)
            fact4=TransAm(np,n)*UintTM(m,np)*fact4
            sum2=sum2+fact3+fact4
 261      continue
          CscaTM(m,n)=dpsi_o(n)*ccc*ba(n)
          CscaTM(m,n)=(sum1-CscaTM(m,n))/dzeta_o(n)
          DscaTM(m,n)=ccc*aa(n)*psi_o(n)
          DscaTM(m,n)=(sum2-DscaTM(m,n))/zeta_o(n)
c       print*,n,CscaTM(m,n),DscaTM(m,n)
 251    continue

c ..................................................................................
c . We can now calculate the Efficiencies (Qext, Qsca, Qabs) .
c ..................................................................................
          do 510 n=1,nbg
            tau=CscaTE(m,n)*dconjg(CscaTE(m,n))
            tau=tau+DscaTE(m,n)*dconjg(DscaTE(m,n))
            pie=CscaTM(m,n)*dconjg(CscaTM(m,n))
            pie=pie+DscaTM(m,n)*dconjg(DscaTM(m,n))
            x=n*(n+1.)*(tau+pie)

            s1=CscaTE(m,n)*dconjg(aa(n))
            s1=s1+DscaTE(m,n)*dconjg(ba(n))
            s2=CscaTM(m,n)*dconjg(ccc*ba(n))
            s2=s2+DscaTM(m,n)*dconjg(ccc*aa(n))
            s3=n*(n+1.)*(s2+s1)
            if(m.eq.0) then
              x=2.0*x
              s3=-2.0*s3
            else
              x=4.0*x
              s3=-4.0*s3
            endif

            qsca=qsca +x
            qext=qext +dreal(s3)
 510      continue
c ......................................................................................................
c .  That finishes one loop in m, now we move to the next m and repeat     .
c .  the same procedure all over again, storing our internal results        .
c .  into T_te, U_te, T_tm, U_tm, and storing the scattering results        .
c .  into C_te, D_te, C_tm, D_tm.                                            .
c ......................................................................................................
 299  continue
c......................................................................
c DONE!!  The sum over m is now complete!!! .
c......................................................................
```

```
      qext=qext/cdabs(x0)**2
      qsca=qsca/cdabs(x0)**2
      qabs=qext-qsca
c     write(8,*) real(distance),real(qext),real(qsca),real(qabs)

      print*,'Qext, Qsca, Qabs=',qext,qsca,qabs
c ************************************************************
c COMMENT THIS BLOCK OUT IF YOU DON'T WANT TO FIND
c THE ASYMMETRY PARAMETER.  THIS FIND_G SUBROUTINE
c IS VERY TIME CONSUMING
      ii=isize1
      alpha=inc_ang
c     gg=find_g(nbg,ii,alpha,CscaTE,DscaTE,CscaTM,DscaTM)
      gg=gg/x0**2/qsca
      print*, 'asym parameter =',gg
c ************************************************************

      print*, '**************************************************'
      print*, 'Done with the rough stuff, now to find the Mueller elements'
      print*, '**************************************************'

c ..........................................................................
c .  Here's the meat of the problem.  We will calculate the amplitude      .
c . scattering matrices for the fields that we've found.  The angle        .
c . theta is our observation angle.  Num is the number of angles we        .
c . want in determining the scattering intensity.  The rest of this        .
c . should look almost like the Mie Code found in Bohren and Huffman       .
c . We have here 3 nested loops.  The first one is to determine the        .
c . angles we want to look at, the second one is the sum over m, and       .
c . the third one sums over n.                                             .
c . If we want to have the observation angle's resolution of 1 degree     .
c . then set num=360, if we want finer resolution, let num be a bigger     .
c . number such as 720 (to have resolution of 0.5degrees)                 .
c ..........................................................................

       phi_o=dfloat(phi*pi/180.0)
   i=0
   d_ang=360.0/num
       b_angle=inc_ang+180.
       f_angle=inc_ang
       do 399 j=1,num+1
     angle=d_ang*(j-1)
       angledd=angle
       if((angle.le.180.0).and.(angle.ge.0.0)) then
          print*,'Obs ANGLE IS ',angle
```

```
            if(angle.gt.180.0) then
               angle=360.0-angle
               phi=pi + phi_o
            else
               phi=phi_o
            endif
           i=i+1
         s1TE(i)=0.
         s3TE(i)=0.
         s2TM(i)=0.
         s4TM(i)=0.

         theta=angle*pi/180.
         x=dcos(theta)

         do 330 m=0,nbg
c        .........................................................................
c        . The next few lines will calc the Legendre polynomial.
c        . at the angle theta                                              .
c        .........................................................................
             ii=nbg+1
             mmm=m
             call nplgndr(mmm,ii,isize1,legpol2,x)
             mmm=m+1
             call nplgndr(mmm,ii,isize1,legpol1,x)
             if (m.eq.0) then
                do 305 jj=-2,isize1
                   legpol(jj)=-legpol1(jj)
305               continue
             else
                mmm=m-1
                call nplgndr(mmm,ii,isize1,legpol,x)
             endif

             do 310 n=m,nbg
c        ..........................................................
c      . Calculate the corresponding pie and tau .
c        ..........................................................
               tau=sqrt((n+m+1.)*(n-m))*legpol1(n)
               pie=tau
               tau=-(tau-sqrt((n+m)*(n-m+1.))*legpol(n))/2.
               if (dabs(x).gt.0.5) then
                 pie=pie+sqrt((n+m)*(n-m+1.))*legpol(n)
                 pie=pie*.5/x
               else
```

```
                 pie=m*legpol2(n)/dsin(theta)
                endif
c ..........................................................................
c .Done with finding Tau and Pie!                                          .
c .Now to find the scattering amplitude S1,S2,S3,and S4 .
c ..........................................................................

                ifact=(-ccc)**n
                      if(m.eq.0) then
                        fact1=DscaTE(0,n)*pie+CscaTE(0,n)*tau
                        fact2=CscaTM(0,n)*pie+DscaTM(0,n)*tau
                        fact3=CscaTE(0,n)*pie+DscaTE(0,n)*tau
                        fact4=DscaTM(0,n)*pie+CscaTM(0,n)*tau
                        s1=ifact*fact1
                        s2=ifact*fact2
                        s3=ifact*fact3
                        s4=ifact*fact4
                      else
                  fact1=DscaTE(m,n)*pie+CscaTE(m,n)*tau
                        fact1=fact1*2.0*dcos(m*phi)
                  fact2=CscaTM(m,n)*pie+DscaTM(m,n)*tau
                        fact2=fact2*2.0*dcos(m*phi)
                        fact3=CscaTE(m,n)*pie+DscaTE(m,n)*tau
                        fact3=fact3*2.0*ccc*dsin(m*phi)
                        fact4=DscaTM(m,n)*pie+CscaTM(m,n)*tau
                        fact4=fact4*2.0*ccc*dsin(m*phi)
                        s1=ifact*fact1
                        s2=ifact*fact2
                        s3=ifact*fact3
                        s4=ifact*fact4
                      endif
                s1TE(i)=s1TE(i)+s1
                s2TM(i)=s2TM(i)+s2
                      s3TE(i)=s3TE(i)+s3
                      s4TM(i)=s4TM(i)+s4
 310              continue
c ...................................................................
c . Finished summing over n, now need to       .
c . sum over m in order to have the complete .
c . solution for the Scat Amp Matrix            .
c ...................................................................
 330        continue

                  s1TE(i)=s1TE(i)
                  s2TM(i)=-ccc*s2TM(i)
```

```
               s3TE(i)=-ccc*s3TE(i)
               s4TM(i)=s4TM(i)
c ......................................................................
c . Done summing over m.  Now we have one         .
c . complete set of scat amp S(i)                 .
c ......................................................................
c ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
c .  Lastly, we can now use the scattering amplitude matrix
c .  and solve for the Mueller Matrix elements.  Once we have
c .  the MM, we have all the informations we need.
c ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
      do 410 jj=1,4
        mm(1,jj)=0.0
        mm(2,jj)=0.0
        mm(3,jj)=0.0
        mm(4,jj)=0.0
 410  continue
      fact1=s1TE(i)*dconjg(s1TE(i))/2.
        fact2=s2TM(i)*dconjg(s2TM(i))/2.
        fact3=s3TE(i)*dconjg(s3TE(i))/2.
        fact4=s4TM(i)*dconjg(s4TM(i))/2.
        c=s2TM(i)*dconjg(s3TE(i))
        cc=s1TE(i)*dconjg(s4TM(i))
        mm(1,1)=dreal(fact1+fact2+fact3+fact4)
        mm(1,2)=dreal(fact2-fact1-fact3+fact4)/mm(1,1)
        mm(1,3)=dreal(c + cc)/mm(1,1)
        mm(1,4)=dimag(c -cc)/mm(1,1)

        mm(2,1)=dreal(fact2-fact1+fact3-fact4)
        mm(2,2)=dreal(fact2+fact1-fact3-fact4)/mm(1,1)
        mm(2,3)=dreal(c -cc)/mm(1,1)
        mm(2,4)=dimag(c+cc)/mm(1,1)

        fact1=s2TM(i)*dconjg(s4TM(i))
        fact2=s1TE(i)*dconjg(s3TE(i))
        fact3=s1TE(i)*dconjg(s2TM(i))
        fact4=s3TE(i)*dconjg(s4TM(i))
        c=s2TM(i)*dconjg(s1TE(i))
        cc=s4TM(i)*dconjg(s3TE(i))
        mm(3,1)=dreal(fact1+fact2)
        mm(3,2)=dreal(fact1-fact2)/mm(1,1)
        mm(3,3)=dreal(fact3+fact4)/mm(1,1)
        mm(3,4)=dimag(c + cc)/mm(1,1)

        fact1=s4TM(i)*dconjg(s2TM(i))
```

```
          fact3=s1TE(i)*dconjg(s2TM(i))
          mm(4,1)=dimag(fact1+fact2)
          mm(4,2)=dimag(fact1-fact2)/mm(1,1)
          mm(4,3)=dimag(fact3-fact4)/mm(1,1)
          mm(4,4)=dreal(fact3-fact4)/mm(1,1)
       print*,real(angledd),real(mm(1,1))
       write(8,*)int(angle),real(mm(1,1)),real(mm(1,2)),
     + real(mm(3,3)),real(mm(3,4))
      endif
c ----------------------------------------------------------------------
c  This endif statement lets us select the angles we want to look at  .
c ----------------------------------------------------------------------
 399  continue
c        .......................................................................
c        . Done with calculating the amplitude coefficients    .
c        .......................................................................

       print*, 'That is all, Folks! '
 999   close(4)
      close(8)
      stop
      end
```

This is the Input File

0.6328	The free space wavelength (wavel)
0.525000	The host radius in microns (rad1)
0.125	radius of the inclusion (rad2)
0.1000	center-center separation distance (note: d+rad2 < rad1)
1.550 0.0d-1	refractive index of host (m1)
1.550 0.0d-1	refractive index of inclusion (m2)
0.0	incident angle of beam (inc_ang)
20	number of obs. angles between 0 and 360 (num)
0.0	angle phi (relative to plane)

```
c       .........................................................................
c       .                         DECLARE.DEF                                  .
c       .                                                                      .
c       . This is the declaration for the variables which will be              .
c       . used in the scattering program.  All the arrays and other            .
c       . variables can be declared and changed in here.                       .
c       .........................................................................

c The parameter *isize* is the physical dimension of the arrays.  We
c should make it at least as big as nbg (suggest nbg+2).  Also,
c you should make sure that the parameter nmax in the subroutines
c mprove and ludcmp should be at least 2*isize. If you don't care
c about the scattering matrix elements, set isiza=1; otherwise, set
c isiza=361 (one for each degree from 0 to 360 degrees).

        parameter (isize=200, isiza=361)
        parameter (isize1=isize+1)
        parameter (isiz2=2*isize, isiz4=4*isize)

c For the MEDIUM, which in our case is just plain old air.
        double precision besj_o(-1:isize),temp(-1:isize)
        double precision psi_o(-1:isize).dpsi_o(-1:isize)
        complex *16 hankel_o(-1:isize),tempc(-1:isize)
        complex *16 zeta_o(-1:isize),dzeta_o(-1:isize)

c For the HOST SPHERE, all the arrays need be at least nbg
        complex *16 hankel1_1(-1:isize)
        complex *16 zeta1_1(-1:isize),dzeta1_1(-1:isize)
        complex *16 hankel2_1(-1:isize)
        complex *16 zeta2_1(-1:isize),dzeta2_1(-1:isize)

c For the INCLUSION SPHERE
        complex *16 besj_2(-1:isize)
        complex *16 psi_2(-1:isize),dpsi_2(-1:isize)
        complex *16 hankel1_2(-1:isize)
        complex *16 zeta1_2(-1:isize),dzeta1_2(-1:isize)
        complex *16 hankel2_2(-1:isize)
        complex *16 zeta2_2(-1:isize),dzeta2_2(-1:isize)

c For the separation, and besj_d need be at least 4*nbg
        complex *16 xd, besj_d(-1:isiz4)

c For the refractive indices and k-vectors
        double precision m_r1,m_r2,m_i1,m_i2
        complex *16 k0,k1,k2,x0,x1,x2,x3,m_1,m_2
```

```
c For the Internal Coefficients
        complex *16 TintTE(0:isize1,0:isize1), UintTE(0:isize1,0:isize1)
        complex *16 TintTM(0:isize1,0:isize1), UintTM(0:isize1,0:isize1)

c For the SCATTERING
        complex *16 CscaTE(0:isize1,0:isize1),DscaTE(0:isize1,0:isize1)
        complex *16 CscaTM(0:isize1,0:isize1),DscaTM(0:isize1,0:isize1)
        complex *16 Q_r(0:isize),Q_s(0:isize)
        complex *16 s1TE(isiza),s2TM(isiza),s3TE(isiza),s4TM(isiza)
        complex *16 s1,s2,s3,s4,sum1,sum2,fact1,fact2,fact3,fact4

c For the NORMALIZED ASSOCIATED LEGENDRE POLYNOMIALS
        double precision legpol(-2:isize1),legpol1(-2:isize1)
        double precision legpol2(-2:isize1), tau,pie
        double precision inc_ang,x

c For the TRANSLATION COEFFICIENTS
        complex *16 cuplo(-1:isiz4), cuplo1(-1:isiz4)
        complex *16 cuplom(0:isiz2,0:isiz2), cuplom1(0:isiz2,0:isiz2)
        complex *16 TransA,TransB,TransAm(0:isize,0:isize)
        complex *16 TransBm(0:isize,0:isize)
        complex *16 c,cc,ccc,ifact

c For the MATRICES IN THE LU_DEC0MPOSTION
        dimension indx(isiz2)
        complex *16 matrix(isiz2,isiz2), matrixlu(isiz2,isiz2)
        complex *16 b(isiz2),bd(isiz2),aa(0:isize),ba(0:isize)

c MUELLER MATRIX
        double precision mm(4,4)
```

```
c        *********************************************************
c        ********** THESE ARE THE SUBROUTINES *****************
c        *********************************************************

      subroutine bessel(n,np,bessj,x)
c ..........................................................................................
c .subroutine to calculate spherical bessel functions of the first        .
c .kind from order -1 to order n, the result is stored in the array       .
c .bessj.  n is related to the size parameter (nbg), and np is the        .
c .physical dimension of the bessj array (isize).  x is a real argument .
c ..........................................................................................

      implicit double precision (a-h,o-z)
      parameter(iacc=500,bigno=1d100,bigni=1d-100)
      integer n,np
      double precision bessj(-1:np),x,xx,pi
      double precision bjp,bj,factor,ax

      pi=1.0
      pi=4.*datan(pi)
      xx=x-2*pi*int(x/2./pi)
      if (n.lt.3) then
          n=3
      endif
c ..........................................................................................
c . If x is less than n, we must use the downward recurrence relation,  .
c . otherwise our results for bessj will be unstable.  In fact, we will    .
c . use the downward alot more than the upward because our x will       .
c . often be less than n, i.e. the size parameter is less than the order    .
c ..........................................................................................
      if (x.lt.n) then
      ax=dabs(x)
      axx=x-2*pi*int(ax/2./pi)
        if (ax.eq.0) then
          bessj(0)=1.
          do 10 i=1, n
              bessj(i)=0.
 10       continue
        else
          m=2*((n+int(sqrt(float(iacc*n))))/2)
c ...............................................................................
c . m is the highest order to calculate; the higher m is   .
c . the more accuracy we'll get, but we'll keep n terms   .
c ...............................................................................
```

```
        do 15 j=0,n
            bessj(j)=0.
 15       continue
        bjp=0.
        bj=1.
        do 20 j=m,1,-1
            bjm=(2.*j+1.)/ax*bj-bjp
            bjp=bj
            bj=bjm
            if(dabs(bj).gt.bigno) then
c ..........................................................................
c . If bj is huge, we must renormalize to prevent overflow .
c ..........................................................................
                bj=bj*bigni
                bjp=bjp*bigni
                do 17 i=0,n
                    bessj(i)=bessj(i)*bigni
 17               continue
            endif
            if (j.le.n) bessj(j)=bjp
 20       continue
        bessj(0)=bj
          sum=0.
          do 25 j=0,n
              sum=sum+(2.*j+1.)*bessj(j)*bessj(j)
 25       continue
        do 30 j=0,n
            bessj(j)=bessj(j)/dsqrt(sum)
            if (x.lt.0.and.mod(n,2).eq.1) bessj(j)=-bessj(j)
 30      continue
        bessj(-1)=dcos(xx)/x
      endif
    else
c ......................................................................
c . Use upward recurrence for when x is greater than n .
c ......................................................................
        bessj(-1)=dcos(xx)/x
        bessj(0)=dsin(xx)/x
        do 40 i=0,n-1
           bessj(i+1)=bessj(i)*(2.*i+1.)/x-bessj(i-1)
 40       continue
    endif
    return
    end
```

```
c       *************************************************
     subroutine c_bessel(n,np,bessj,x)
c .........................................................................
c .subroutine to calculate complex spherical bessel functions of the  .
c .first kind from order -1 to order n.  The result is stored in array  .
c .bessj.  This is just like the bessel function, except we now have   .
c .complex arguments, i.e. x is a complex variable                      .
c .........................................................................

     parameter(iacc=500,bigno=1d100,bigni=1d-100)

        implicit double precision (a-h,o-z)
     double precision pi,xx
     integer n,np
     complex *16 x,xxxx,sum
     complex *16 bessj(-1:np)
     complex *16 bjp,bj,factor,bjm

     if (n.lt.3) then
        n=3
     endif

     xx=cdabs(x)
     pi=1.0
     pi=4.*datan(pi)
     xxxx=x
     if (xx.gt.32760) then
         xxxx=x-2*pi*int(x/2./pi)
     endif

     if (xx.lt.n) then
c ..........................................
c . Use downward recurrence  .
c ..........................................
       if (xx.eq.0) then
         bessj(0)=1.
         do 10 i=1,n
             bessj(i)=0.
 10        continue
       else
         m=2*((n+int(sqrt(float(iacc*n))))/2)
         do 15 j=0,n
             bessj(j)=0.
 15        continue
         bjp=dcmplx(0.d0,0.)
        bj=dcmplx(1.d0,1.)
        do 20 j=m,1,-1
           bjm=(2.*j+1.)/x*bj-bjp
           bjp=bj
           bj=bjm
           xxx=cdabs(bj)
           if(xxx.gt.bigno) then
c ...................................................
c . Renormalize to prevent overflow  .
c ...................................................
              bj=bj*bigni
              bjp=bjp*bigni
              do 17 i=0,n
                 bessj(i)=bessj(i)*bigni
 17           continue
           endif
           if (j.le.n) bessj(j)=bjp
 20     continue
        bessj(0)=bj
          sum=0.
          do 25 j=0,n
             sum=sum+(2.*j+1.)*bessj(j)*bessj(j)
 25       continue
        do 30 j=0,n
           bessj(j)=bessj(j)/cdsqrt(sum)
 30     continue
        bessj(-1)=cdcos(xxxx)/x
      endif

    else
c ........................................
c . Use upward recurrence  .
c ........................................
        bessj(-1)=cdcos(xxxx)/x
        bessj(0)=cdsin(xxxx)/x
        do 40 i=0,n-1
          bessj(i+1)=bessj(i)*(2.*i+1)/x-bessj(i-1)
 40     continue
    endif
    return
    end
```

```
c        ****************************************************
     subroutine newman(n,np,bessy,x)
c ..........................................................................
c .  subroutine to calculate the spherical bessel function of the       .
c .  second kind (or Newmann function) from order -1                    .
c .  to order n; result stored in array bessy.  bess_y  is the common  .
c .  notation for the Newmann function.                                 .
c ..........................................................................

         implicit double precision (a-h,o-z)
     integer n,np
     double precision x,xx,pi
     double precision bessy(-1:np)

     if (n.lt.3) then
          n=3
     endif
     pi=1.0
     pi=4.*datan(pi)
     xx=x-2.*pi*int(x/2./pi)
c ......................................................................
c .Must use this xx because it's a value between 0 and 2pi. and it .
c .works as the argument for the sin and cos; if we just stuck in   .
c .x instead, it won't work.                                          .
c ......................................................................
     bessy(-1)=dsin(xx)/x
     bessy(0)=-dcos(xx)/x
     bessy(1)=-dcos(xx)/x/x-dsin(xx)/x
     do 10 i=2,n
          bessy(i)=(2.*(i-1.)+1.)/x*bessy(i-1)-bessy(i-2)
 10   continue

     return
     end

c        ****************************************************

     subroutine c_newman(n,np,bessy,x)
c ..........................................................................
c . subroutine to calculate the complex spherical bessel function     .
c . from order 0 to order n, the result is stored in array bessy.      .
c . This is the Newman function, aka, bessel function of order 2.     .
c ..........................................................................

          implicit double precision(a-h,o-z)
```

```
      integer n,np
      complex *16 x,xx,bessy(-1:np)
      double precision pi

      if (n.lt.3) then
         n=3
      endif
      pi=1.0
      pi=4.*datan(pi)
      xx=x-2.*pi*int(x/2./pi)
      bessy(-1)=cdsin(xx)/x
      bessy(0)=-cdcos(xx)/x
      bessy(1)=-cdcos(xx)/x/x-cdsin(xx)/x
      do 10 i=2,n
         bessy(i)=(2.*(i-1.)+1.)/x*bessy(i-1)-bessy(i-2)
 10   continue
      return
      end

c        **************************************************

      subroutine nplgndr(m,nbr,iisize,legpol,x)
c ...........................................................................
c . Calculate normalized ass. legendre pol. from l=0 to nbr using value of   .
c . -1<=x<=1   Note that everything here is double precision because other .
c . wise we've got a runaway solution.                                      .
c ...........................................................................

      implicit double precision (a-h,o-z)
      double precision x
      integer m,nbr,iisize
      double precision legpol(-2:iisize)
      double precision pmm,somx2,fact,pll,pmmp1
      mm=mod(m,2)*(-2)+1
      m=abs(m)
      do 5 i=-2,m
         legpol(i)=0.
 5    continue
      pmm = 1.
      if (m.gt.0) then
         somx2=sqrt((1.-x)*(1.+x))
         fact=1.
         do 11 i=1,m
            pmm=-pmm*fact*somx2/sqrt(real(i))
            fact=fact+2.
```

```
11         continue
        do 110 i=m+1,2*m
            pmm=pmm/sqrt(real(i))
110        continue
        pmm=pmm*sqrt(2.*m+1.)
    endif
    legpol(m)=pmm*mm
    if (nbr.gt.m) then
        pmmp1=x*sqrt(2.*m+3.)*pmm
        legpol(m+1)=pmmp1*mm
    endif
    if (nbr.gt.m+1) then
        do 12 ll=m+2,nbr
            pll=x*sqrt(2.*ll-1.)*pmmp1
            pll=pll-sqrt((ll+m-1.)*(ll-m-1.)/(2.*ll-3.))*pmm
            pll=pll/sqrt((ll-m)*(ll+m)/(2.*ll+1.))
            pmm=pmmp1
            pmmp1=pll
            legpol(ll)=pll*mm
12         continue
    endif
    m=m*mm
    return
    end


c       ***************************************************

        subroutine riccati(nbg,hval,zeta,dzeta,x)
        integer nbg,i
        complex *16 hval(-1:nbg),zeta(-1:nbg),dzeta(-1:nbg),x

        zeta(-1)=x*hval(-1)
        do 10 i=0,nbg
                zeta(i)=x*hval(i)
                dzeta(i)=x*hval(i-1) -i*hval(i)
10      continue
        return
        end
```

```
c       *************************************************
        subroutine ludcmp(a,n,np,indx,d)
c .......................................................................
c . given nxn matrix a with physical dim NP, this routine replaces it    .
c . by the LU decomposition of a rowwise permutation of itself.  A and   .
c . n are input. A is output. INDX is an output vector which records the .
c . row permutation effected by the partial pivoting; D is output        .
c . as +-1 depending on whether the number of row interchanges           .
c . was even or odd.  This routine is used in combination with Lubksb    .
c . to solve linear equations or invert a matrix.                        .
c .......................................................................

        complex *16 tiny
        parameter (nmax=150)
c ................................
c . this nmax value should be at  .
c . least twice the value of nbg  .
c ................................
        complex *16 a(np,np),sum,cdum,vv(nmax),aamax
        dimension indx(n)

        tiny=dcmplx(1.0d-80,1.0d-80)
        d=1.
        do 12 i=1,n
           aamax=dcmplx(0.d0,0.)
           do 11 j=1,n
              if (cdabs(a(i,j)).gt.cdabs(aamax)) aamax=a(i,j)
 11        continue
           if (cdabs(aamax).eq.0) pause 'singular matrix'
           vv(i)=1./aamax
 12     continue
        do 19 j=1,n
           do 14 i=1,j-1
              sum=a(i,j)
              do 13 k=1,i-1
                 sum=sum-a(i,k)*a(k,j)
 13           continue
              a(i,j)=sum
 14        continue
           aamax=dcmplx(0.d0,0.)
           do 16 i=j,n
              sum=a(i,j)
              do 15 k=1,j-1
                 sum=sum-a(i,k)*a(k,j)
 15           continue
```

```
            a(i,j)=sum
            cdum=vv(i)*sum
            if (cdabs(cdum).ge.cdabs(aamax)) then
                imax=i
                aamax=cdum
            endif
16        continue
          if (j.ne.imax) then
            do 17 k=1,n
                cdum=a(imax,k)
                a(imax,k)=a(j,k)
                a(j,k)=cdum
17            continue
            d=-d
            vv(imax)=vv(j)
          endif
          indx(j)=imax
          if(cdabs(a(j,j)).eq.0) a(j,j)=tiny
          if (j.ne.n) then
            cdum=1./a(j,j)
            do 18 i=j+1,n
                a(i,j)=a(i,j)*cdum
18            continue
          endif
19    continue
      return
      end

c         **************************************************

      subroutine lubksb(a,n,np,indx,b)
c .....................................................................
c . Solves the set of n linear equations A.X=B. Here A is input, not   .
c . as the matrix A but rather as its LU decomp, from LUDCMP.  INDX    .
c . is input as the permutation vector returned by LUDCMP. B is input  .
c . as the right-hand side vector B, and returns with the solution     .
c . Vector X.  A,N,NP and INDX are not modified by this routine and    .
c . can be left in place for successive calls with different right-hand .
c . sides B.  This routine takes into account the possibility that B   .
c . can begin with many zero elements, so it is efficient for use in   .
c . matrix inversion.                                                  .
c .....................................................................

       complex *16 a(np,np),b(n),sum
       dimension indx(n)
      ii=0
      do 12 i=1,n
          ll=indx(i)
          sum=b(ll)
          b(ll)=b(i)
          if (ii.ne.0) then
              do 11 j=ii,i-1
                  sum=sum-a(i,j)*b(j)
11             continue
          else if (cdabs(sum).ne.0.) then
              ii=i
          endif
          b(i)=sum
12    continue
      do 14 i=n,1,-1
          sum=b(i)
          if (i.lt.n) then
              do 13 j=i+1,n
                  sum=sum-a(i,j)*b(j)
13             continue
          endif
          b(i)=sum/a(i,i)
14    continue
      return
      end

c        **************************************************

      subroutine mprove(a,alud,n,np,indx,b,x)
c ...............................................................................
c . improves a solution vector X of the linear set of equations A*X=B  .
c . The matrix A, and the vectors B and X are input, as is the dim     .
c . n.  Also input is alud the ludcomp of A as returned by ludcmp, and .
c . the vector indx also returned by that routine.  On output, only    .
c . X is modified, to an improved set of values.                       .
c ...............................................................................

      parameter (nmax=150)
c ...........................................................
c . all the values here should be the same as those      .
c . found in the ludcmp subroutine, eg nmax should  .
c . have the same numerical value as in ludcmp          .
c ...........................................................
      complex *16 a(np,np),alud(np,np),b(n),x(n),sdp,r(nmax)
      dimension indx(n)
```

```
   do 12 i=1,n
       sdp=-b(i)
       do 11 j=1,n
           sdp=sdp+a(i,j)*x(j)
11       continue
       r(i)=sdp
12   continue
    call lubksb(alud,n,np,indx,r)
    do 13 i=1,n
        x(i)=x(i)-r(i)
13   continue
    return
    end

c        *************************************************

         double precision function find_g(nbg,nn,ang,cte,dte,ctm,dtm)
c ..............................................................................
c . This function will calculate the asymmetric parameter g which is .
c . used in determining radiative transfer.  It will call the subrou-    .
c . tine "rotation" and use the resulting values to calculate g          .
c . Passed into this function are the values for nbg, nn = isize,        .
c . the incident angle ang, and the 4 scattering amplitude coefficients .
c ..............................................................................
         integer nbg,nn
         double precision ang
         complex *16 cte(0:nn,0:nn),dte(0:nn,0:nn)
         complex *16 ctm(0:nn,0:nn),dtm(0:nn,0:nn)
         parameter (size=76)
c ..........................................................
c . if nbg is bigger than this size value,      .
c . then change size to equal at least nbg.  .
c ..........................................................
         double precision t1,t2,fact,fn,asym_g,gg
         complex *16 a(-size:size,0:size),b(-size:size,0:size),cc
         complex *16 c(-size:size,0:size),d(-size:size,0:size),ccc
         integer n,m

         gg = 0.0
         ccc=dcmplx(0.0d0,1.0)
         ij=size
c        print*,'nn,ij=',nn,ij
         call rotation(a,b,c,d,nbg,ang,cte,dte,ctm,dtm,nn,ij)
         do 10 n=0,nbg,1
```

```
      do 20 mp=-n,n,1
        m=mp
        t1=m*dreal(a(m,n)*dconjg(b(m,n)) +c(m,n)*dconjg(d(m,n)))
        fn=dfloat(n)
        fact=(fn-m+1.)*(fn+m+1.)/(2.*fn+3.)/(2.*fn+1.)
        fact=fn*(fn+2.)*dsqrt(fact)
        cc=a(m,n)*dconjg(a(m,n+1))+b(m,n)*dconjg(b(m,n+1))
        cc=cc+c(m,n)*dconjg(c(m,n+1))+d(m,n)*dconjg(d(m,n+1))
        t2=fact*dreal(ccc*cc)
        asym_g=4.*(t1+t2)
        gg = gg+asym_g
 20   continue
 10   continue
      find_g=gg
      return
      end

c     ************************************************

      subroutine rotation(a,b,c,d,nbg,ang,cte,dte,ctm,dtm,nn,ij)
c ..........................................................................
c .  This subroutine calculates the new rotated values of the scattering .
c .  coefficients C's and D's, and return these new values to the fuction.
c .  find_g to find the asym parameter g.                                 .
c ..........................................................................
      integer nbg,nn,ij
      complex *16 a(-ij:ij,0:ij),b(-ij:ij,0:ij)
      complex *16 c(-ij:ij,0:ij),d(-ij:ij,0:ij)
      double precision ang
      complex *16 cte(0:nn,0:nn),dte(0:nn,0:nn)
      complex *16 ctm(0:nn,0:nn),dtm(0:nn,0:nn)

      parameter (siz=200)
c ....................................................
c . the parameter siz here must be at least   .
c . twice the parameter size given in find_g .
c ....................................................
      double precision factorial
      double precision beta(-siz:siz,-siz:siz)
      double precision t1,t2,t3,t4,fact,term1,term2
      complex *16 sum1,sum2,sum3,sum4,c1,c2,c3,c4
      integer n,mp,m,ii,jj,i_high,i_low

c     print*,'angle=',ang
      ang=-ang
```

```
        do 10 n=0,nbg
c       print*,'rotation n= ',n
          do 15 m=-n,n
            a(m,n)=cmplx(0.0,0.0)
            b(m,n)=cmplx(0.0,0.0)
            c(m,n)=cmplx(0.0,0.0)
            d(m,n)=cmplx(0.0,0.0)
 15       continue

c ......................................................................
c . For each value of n, we will find all possible values             .
c . of the BETA(m,mp), and find the resulting new scat-               .
c . tering coefficients a,b,c,d                                       .
c ......................................................................
          do 20 mp=-n,n
            do 30 m=-n,n
                t1=factorial(n+mp)
                t2=factorial(n-mp)
                t3=factorial(n+m)
                t4=factorial(n-m)
                term1=t1+t2-t3-t4
                term2=dexp(term1)
                beta(mp,m)=dsqrt(term2)

                sum1=cmplx(0.0,0.0)
                i_low=0
                if((-m-mp).gt.i_low) i_low=-m-mp
                i_high=n-m
                if((n-mp).lt.i_high) i_high=n-mp
                if(i_low.gt.i_high) then
                       beta(mp,m)=0.0
                else
                 do 40 jj=i_low,i_high,1
                  t1=factorial(n+m)
                  t2=factorial(n-mp-jj)
                  t3=factorial(m+mp+jj)
                  t4=t1-t2-t3
                  term1=dexp(t4)

                  t1=factorial(n-m)
                  t2=factorial(jj)
                  t3=factorial(n-m-jj)
                  t4=t1-t2-t3
                  term2=dexp(t4)
                  ii=n-m-jj
                  ii=mod(ii,2)
                  if(ii.eq.0)then
                       fact=1.0
                  else
                       fact=-1.0
                  endif
        if(term1.gt.1.d200.or.term1.lt.1.d-200)then
                print*,'mp,m, beta=',mp,m,beta(mp,m)
                stop
        endif
                  ii=2*jj+mp+m
                  t3=(dcosd(ang/2.))**ii
                  ii=2*n-2*jj-mp-m
                  t4=(dsind(ang/2.))**ii
                  sum1=sum1+(term1*term2)*fact*t3*t4
 40               continue
                endif
                beta(mp,m)=beta(mp,m)*sum1
c ......................................................
c . That's one value of m computed for  .
c . each given value of mp              .
c ......................................................
 30           continue
 20         continue
c .....................................................................
c .  There!  We've just finished calculating all the      .
c .  possible values of BETA(mp,m)  for each given  .
c .  value of n.                                             .
c .....................................................................

c ....................................................................................
c .  Now to find the new values for the scattering              .
c .  coefficients, using the BETA that we've just found        .
c ....................................................................................
          do 50 mp=-n,n,1
            sum1=cmplx(0.0,0.0)
            sum2=cmplx(0.0,0.0)
            sum3=cmplx(0.0,0.0)
            sum4=cmplx(0.0,0.0)
 666        do 60 m=-n,n,1
                if(m.le.0) then
                  ii=abs(m)
                  if(mod(ii,2).eq.0) then
                       c1=cte(ii,n)
```

```
                    c2=-dte(ii,n)
                    c3=-ctm(ii,n)
                    c4=dtm(ii,n)
                 else
                    c1= -cte(ii,n)
                    c2= dte(ii,n)
                    c3= ctm(ii,n)
                    c4= -dtm(ii,n)
                 endif
               else
                 c1=cte(m,n)
                 c2=dte(m,n)
                 c3=ctm(m,n)
                 c4=dtm(m,n)
               endif
               sum1 = sum1+c1*beta(m,mp)
               sum2 = sum2+c2*beta(m,mp)
               sum3 = sum3+c3*beta(m,mp)
               sum4 = sum4+c4*beta(m,mp)
60          continue
c ..........................................................................
c . These a.b.c.d are now the new values of the scattering .
c . coefficients after the rotation                         .
c ..........................................................................

            a(mp,n)=sum1
            b(mp,n)=sum2
            c(mp,n)=sum3
            d(mp,n)=sum4
50        continue
10      continue
        return
        end

c       ****************************************************

        double precision function factorial(n)
        integer n
        integer loop
        double precision sum
c..........................................................................
c. Since n can be a huge number (like 190) we will take .
c. the log of the factorial, so the machine won't blow    .
c. up.   eg. 10! will be log10+log9+log8+...log1          .
c..........................................................................
```

```
      sum=0.0
      if (n.lt.0) then
              print*,'ERROR!  value less than zero!'
              print*,'value=',n
              stop
      endif
      if (n.gt.1) then
        do 10 loop=1,n
              sum=sum + dlog(dfloat(loop))
10      continue
        factorial=sum
      else
        factorial=0.0
      endif
99    return
      end
```